The Temporal Organisation of Documents and Versions: A User-Centred Investigation

---

A thesis
submitted in partial fulfilment
of the requirements for the Degree
of
Doctor of Philosophy
in the
University of Canterbury
by
Michael JasonSmith

Examining Committee

Doctor Steve Jones	Examiner
Professor Alan Dix	Examiner
Associate Professor Andy Cockburn	Supervisor
Associate Professor Tim Bell	Associate Supervisor
Doctor Neville Churcher	Head of Department

University of Canterbury
2006

To my Granny: Phyllis JasonSmith

Abstract

In this thesis a study of computing systems that use time as the primary method of organising electronic documents, and versions of electronic documents, is presented. Such systems should be useful and usable because they exploit people's intuitive understanding of temporal order. In addition, the systems are worthy of investigation because they have received little attention, yet they may provide enormous benefits to users with little cost, as the temporal information is easy for systems to record. Temporal document-organisation systems have the potential of alleviating many of the problems that traditional systems have had for decades.

Throughout this thesis, user-interface guidelines for the implementation of temporal document-organisation systems are presented. The guidelines are based on empirical and theoretical evaluations that I have conducted, and studies of other's work. By using theses guidelines, designers should be able to create interfaces that are liked by users, and provide good support for the user's tasks.

The first set of guidelines are based on an investigation of the human-factors of temporal document-organisation, specifically looking at memory and temporal awareness. These human factors are related to the user's tasks with documents and document versions: finding, reminding, error-recovery and system exploration. Another set of guidelines, based on a study of how existing document-organisation systems support the user's tasks, are then presented.

My first empirical evaluation looks at history lists, which are some of the most common temporal document-organisation interfaces that are found today. In the study it was found that participants are slower at retrieving Web pages when using an interface that broke the history into non-temporal categories than with the other three interfaces that were tested. In addition the participants preferred the interface that broke the history into 'temporal chunks'.

Following on from the history-list evaluation, a theoretical and empirical evaluation of version retrieval systems, including undo, is presented. It was found that, in a text-editing environment, there is sufficient mechanical rea-

son for forward error-correction to be favoured over undo when correcting small and simple errors. For more complex errors, it was found that a visualisation of the prior document versions is better than forward error-correction and undo. In a similar evaluation of error recovery in a drawing editor, undo was found to be the quickest method of recovering from simple errors, while a visualisation of the prior document versions allowed for faster recovery from more complex errors.

Having looked at the retrieval of documents and document-versions separately, my final study looks at a system that combines them both. The system that I developed organises documents without the need for file names and folders, which are used in most document organisation systems. In the formative study I found that the system that combines the retrieval of documents and document versions is useful and usable, and the organisation of the data did not confuse the participants.

After each of the evaluations, I provide guidelines that should be applicable not only to the interfaces that were studied, but to temporal document-organisation interfaces in general.

Table of Contents

Acknowledgments

I would like to thank my supervisor, Andy Cockburn, for all the help he has given me over the years. Thanks must also go to Michele Bannister, Gina Bayne, Annabel Church, Rebecca Cox, Matt and Jen Walker, and Tim Wright for their support and lunchtime distractions. As always, my parents have provided me with enormous support, for which I could not thank them enough. Finally, all the participants in my evaluations have my sincere gratitude.

Chapter I

Introduction

In this thesis a study of temporal document-organisation and version organisation systems is presented. These systems use temporal properties of electronic documents and versions as the primary organisation method. They are effective because documents and versions can be automatically associated with a date, time is a good cue to retrieval, documents can act as reminders of the user's current tasks, and temporal ordering can be applied to all documents, versions and actions in a system.

There are two problems with common document-organisation systems. The first is the demand of associating metadata with a document, such as a name of the document, the folder that stores the document and the authors that worked on the document. If care is not taken when creating metadata then documents can be difficult to find (Smith et al., 1982; Jacsó, 2005). The second problem with traditional document-organisation systems is that prior versions of documents are destroyed, so they cannot be retrieved in order to correct errors.

Temporally-ordered documents are found in many applications, such as the back button in Web browsers and the messages stored in email clients. As mentioned previously, there are four reasons that such systems should provide good support for the user's tasks, without the drawbacks of traditional systems.

1. The user does not have to manually associate a document with a date, as this can be done automatically by the computing system. This contrasts with file-names or keywords, which often have to be supplied by the user.

2. Time is an effective cue to document retrieval, as people are good at recalling the order of events. When a document is named and placed

in a folder, care must be taken so the chosen name and location will be obvious in the future, otherwise locating the document may be difficult.

3. Temporally ordering documents allows them to act as reminders of the user's current tasks. Documents placed in folders can be hidden from the user's view, so cannot remind the user of the current tasks.

4. Time is pervasive, so every action the user makes, such as creating a document, happens at a particular time, and actions can be distinguished by time.

In this thesis, using empirical and theoretical techniques, I show that using time to organise documents and versions is both useful and usable.

There is a single question that motivates my research: what is the best way to organise documents? This question is large and complex — encompassing research into desk organisation, work-flow, library science, databases, existing computer file systems, human memory and the relationships between people and documents. This question is too large for a single thesis, so most of my research concentrates on one method of document organisation: using time. In particular, I sought initial answers to three questions. First, what is the best way to use time to organise different documents? The second, related, question is what is the best way to use time to organise different *versions* of the same document? There are a lot of similarities between the retrieval of documents and the retrieval of document versions, so my final research question is whether it is possible to combine document and version organisation in the same interface, and will this be useful?

The research questions are explored in two separate, but related, themes that run through the majority of my thesis. The first is the retrieval of documents that are organised by time. As part of this theme, I look at the tasks with documents, systems that organise documents by time, and effective ways to present temporally ordered documents. My aim is to determine, at least partially, the effective methods of presenting temporally ordered documents. The second theme covers the retrieval of document versions that are organised by time. My research in this area follows a similar pattern to the research into documents: I look at the tasks with document versions, systems

that organise versions by time, and the most effective way to present temporally ordered versions. I hope that the research I present will allow me to determine some effective means of organising document versions by time. As I will show, versions of documents are often organised using quite different systems to the documents themselves; I also treat documents and versions separately in Chapters 2–5. However, in Chapter 6 the two themes converge: I present a system that uses the same interface to organise documents and versions of documents by time. The intention is to provide insight into the last of my research questions.

1.1 Area of Research

In this thesis, I present research that can be used to inform the design of interactive computing systems, temporal and non-temporal alike. I use a range of research techniques, common to human-computer interaction (HCI): the area of computer science "concerned with the design, evaluation and implementation of interactive computing systems" (Hewett et al., 1992).

Temporal document-organisation belongs to the sub-field of HCI known as temporal aspects of usability, or TAU. It is not a small field: the research in this area "is characterised by its diversity" (Johnson and Gray, 1996). Research into TAU investigates the temporal proprieties of user interfaces; besides temporal document-organisation, it includes, but is not restricted to, the presentation of documents that have a temporal component (such as videos), people's perception of time and the rate of change, and how the passage of time affects usability and learning. The earliest temporal system I am aware of is an error recovery mechanism that ran on ENIAC in 1948, but recognition of TAU as a field of study is relatively new — the first workshop was in 1995 at the University of Glasgow, which has been running a TAU project since 1992 (Johnson and Gray, 1996; Fabre and Howard, 1998). Recently, Nielsen (2000a) speculated that "history and other time-based attributes" will become very important for user interfaces.

Temporal document-organisation systems use a temporal property of a document, such as the date it was last edited or viewed, to organise documents. Temporal document-organisation systems, and version organisation

systems, are found in most computing environments today. For example, the back button in Web browsers allows the user to return to pages that he or she has seen in the past, undo allows the user to return to prior document versions, and history lists allow documents (which have been recently edited) to be quickly opened. However, most temporal document-organisation systems supplement existing file systems, so the user still has to name, categorise, and save documents.

A temporal document-organisation system that did not rely on a file system could solve many of the problems associated with files and folders. However, while a temporal system should be useful, it is not known how usable such a system would be.

I meet three primary research objectives in this thesis. First, I find out what user-tasks a temporal document-organisation system needs to support, and how to best support those tasks. Next, I establish what usability problems exist in current temporal document-organisation systems, and how to improve them. Finally, I create and evaluate a temporal document-organisation system that does not rely on files. I also meet some minor objectives, which include the creation of guidelines for the design of usable document-organisation interfaces, the creation of a framework that allows for the comparison of document and version organisation-systems, and a summary of the psychological research into the human factors that affect temporal document and version retrieval.

1.2 Research Contributions

To the best of my knowledge, Chapter 2 presents the first survey of research into the area of human factors that affect temporal document-retrieval and version retrieval. As this work is centred on the user, most of the information is drawn from psychology. The guidelines presented in this chapter are used in the user-centred study of systems (Chapter 3), which presents a framework for comparing the support that systems provide for the user's tasks with documents and versions. While other work — such as that by Freeman (1997) — has created taxonomies of document-organisation systems, I present a user-centred view of document and version retrieval, looking at

how the system supports the user's tasks, rather than how the information is stored by the system.

In Chapter 4 I present the results of a controlled empirical evaluation of document retrieval from history lists, which are a type of temporal document-organisation system. While other temporal document-organisation systems have been studied, I am not aware of any other study of the simple and common history-list. I find that retrieval from complex lists, such as that found in Microsoft Internet Explorer, is generally slower than retrievals from simpler lists.

Undo is a system that allows the user to retrieve prior versions of a document, in order to correct errors or as part of system exploration; in Chapter 5 I present a two-part evaluation of undo. The first part is a theoretical study of expert performance using different error-recovery mechanisms, including undo. I show that, in a text editor, correcting the errors by deleting and typing should be faster than undo, but a graphical version of undo should be the fastest method when the errors become large. The theoretical evaluation is then followed by an experiment, which looked at how four error-recovery mechanisms are used in two different domains: text-editing and drawing. The results from the empirical evaluation confirm those from the theoretical study. While there have been many proposals for different types of undo, and much musing about why undo is not used more often, to my knowledge I have conducted the first in-depth evaluation of undo to determine the usability problems with the system.

In Chapter 6 I present the SWACA text editor. It is a system that *only* uses time to organise documents, without the need for document-names and folders. As well as organising documents, SWACA implicitly organises versions of documents, so they can be retrieved as part of system exploration or to recover from errors. I show that the system is useful, and potentially more usable, than many other systems.

A Note on Time

> What, then, is time? As long as no one asks me, I know. As soon as I wish to explain it to him who asks, I know not.
>
> — Saint Augustine of Hippo,
> *Confessions, Book XI, Chapter 14,* AD 397

Time is difficult to define (Priestley, 1964; Nahin, 1998). For the most-part I will rely on an intuitive definition of time, but I wish to clarify some terminology. First, when I refer to 'time' in this thesis, I will be discussing the temporal component of what the physicists call 'space-time', not a particular point in time, such as 07:56 on the 25th day of February, AD 2005. Second, when I refer to a 'date' I will be discussing a point in time, such as the example given before.

Chapter II

Human Factors of Temporal Document Retrieval and Version Retrieval

In this chapter I review four of the human factors that affect temporal document-retrieval and temporal version retrieval: temporal perception, recall from memory, people's tasks with documents, and people's tasks with versions of documents. It is necessary to understand the human factors so that people's limitations and strengths with time and memory are known, the tasks that are carried out with systems are understood, and guidelines for system design can be established. While there are many human factors that affect how a system is used, most are not discussed in this chapter: the cognitive processing speed (Card et al., 1983) and the limits of pre-attentive processing (Healey et al., 1996) are two examples. Instead, I will focus on factors that affect temporal document and version retrieval in particular.

I begin by looking at human temporal awareness, its strengths and limitations (Section 2.1). In Section 2.2 I discuss human memory, as it has an enormous affect on document and version retrieval. These sections fit into all three of the research themes that were introduced in Chapter 1: the retrieval of documents, the retrieval of versions, and the combined retrieval of documents and versions.

In order to better understand what temporal document-retrieval systems should do, I discuss the user's tasks with documents in Section 2.3. It relates to the first of my main research themes: the retrieval of documents that are organised by time. This study leads on to Sections 3.1, 3.2, and Chapters 4 and 6.

Relating to the second of my research themes — the retrieval of document versions — Section 2.4 contains a summary of the research into the user's tasks with versions. This investigation is used in Section 3.3 and Chapters 5

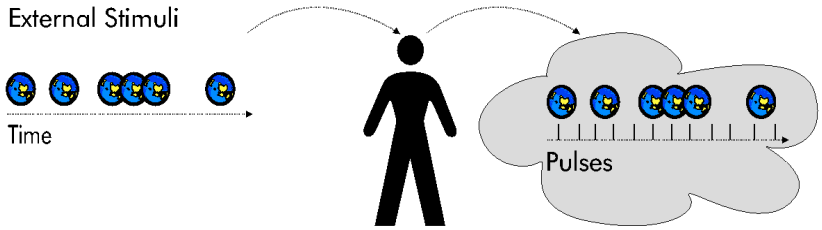


Figure 2.1: The body-clock model of time perception. In this model, external stimuli are matched against 'pulses' that are generated internally.

and 6.

I conclude each of the aforementioned sections with some HCI guidelines, before this chapter is concluded in Section 2.5.

2.1 People and Time

My research concentrates on temporal document retrieval, so it is important to understand how time is perceived by people, and to understand the benefits and limitations of human temporal perception. There are two models for time perception: the body-clock (pulse counter) model, and the information storage (cognitive) model (Michon, 1972). Section 2.1.1 discusses the body-clock model of perception, its failings, and why it was rejected in favour of the information-storage model (Section 2.1.2). Other factors that influence the perception of time are discussed in Section 2.1.3, before the impact of temporal perception on HCI is discussed (Section 2.1.4).

2.1.1 Body-Clock Model

Until the late 1960s the most widely held theory on temporal perception was the body-clock model (Michon, 1972). In this model a person receives external stimuli and matches it to 'pulses' that are generated internally (Figure 2.1). The body-clock model matches how durations are commonly timed in science: with clocks.

The body-clock model became unpopular for two reasons. First, the source of the pulses could not be found: humans have no 'temporal recep-

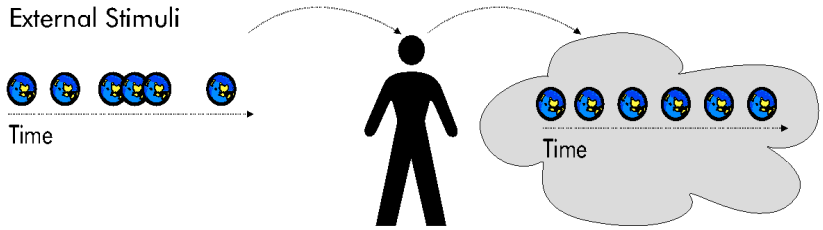


Figure 2.2: Information storage model of time perception. In this model the time between each stimulus is assumed to be constant.

tors' (Brown, 1995). Secondly the body-clock model does not explain the 'filled-duration illusion'. This illusion has the effect of making a period of time that has some stimulus appear *longer* than an equivalent period with no stimulus. The filled-duration illusion has been noted since the early 1950s; many studies have since corroborated the results (Michon, 1972; Hogan, 1978; Brown, 1995). However, it should be noted that people become frustrated when there is *nothing* happening (Miller, 1968; Myers, 1985; Card et al., 1991; Conn, 1995), which implies that a median point exists (Hogan, 1978). The problems with the body-clock model led to the development of the information storage model (Section 2.1.2).

2.1.2 Information-Storage Model

The information-storage model was developed in the late 1960s to provide a model of temporal perception that explained the filled-duration illusion and did not require temporal receptors. The model is based on an information processing view of human cognition (Michon, 1972). In this model an individual perceives stimuli but it is *assumed* that the time between each stimulus is constant (Figure 2.2). This model requires each individual to *remember* what has happened in the past, hence the 'information storage' reference in the model's title. The more the person has to remember the slower time seems, which explains the filled-duration illusion.

Experiments carried out in more recent times, such as those conducted by Brown (1995), confirm these results. There is some debate on whether it is stimulus change or the stimulus itself that an individual remembers, but

this remains undecided.

2.1.3 Other Factors that Influence Time Perception

Besides the presence or absence of stimulus, the other primary factor that alters an individual's perception of time is time itself. Vierordt's Law states that short periods of time are overestimated, while long periods are underestimated (Woodrow, 1951). What constitutes 'short' and 'long' is not clear, as most studies into Vierordt's Law arrive at different values, but the transition appears to be in the 5–20 second range. More recent experiments, such as those conducted by Brown (1995), confirm Vierordt's Law.

Woodrow (1951) lists other experiments carried out in the early part of the twentieth century that involved altering the subjects' perception of time by the use of drugs, pain, and the alteration of body temperature. While interesting to psychologists and psychotherapists, these experiments have little bearing on HCI.

2.1.4 Temporal Perception and HCI

The need for feedback in HCI is well known (Miller, 1968; Myers, 1985; Card et al., 1991; Conn, 1995). Without feedback the user becomes frustrated and the dialogue between the system and the user breaks down. However, the filled-duration illusion (Section 2.1.1) suggests that feedback, such as progress bars and small animations (heartbeats), has the net effect of making computer systems seem *slower* than they are. To lessen the impact of the filled-duration illusion, stimuli should be kept to a minimum. If there is *no* feedback users become frustrated, so some form is required, but if there is a large amount of stimulus provided by the feedback — a quickly moving animation for example — then the system will appear slower than reality.

2.2 People and Recall from Memory

A document retrieval system allows the user to map what he or she recalls about a document onto a retrievable entity. Therefore, understanding how people recall information is important if systems are to be created to support

document retrieval. There are two types of memory from which people recall information: 'short-term memory' (Section 2.2.1) and 'long-term memory' (Section 2.2.2). After discussing both forms of memory, I will examine how the limits and capabilities of human memory affects the temporal aspects of HCI (Section 2.2.3).

2.2.1 Short-Term Memory

Short-term memory allows people to recall a small amount of information, with great accuracy, for a short period of time. The accuracy of recall degrades over time, with most people only able to remember information in short-term memory for 20–30 seconds without rehearsal (McCarthy and Warrington, 1990). In addition, the capacity of short-term memory is also limited — with most people able to remember 7 ± 2 'chunks' (Card et al., 1983). What constitutes a chunk is highly dependent on context, with random sentences, words, letters and digits forming separate chunks (McCarthy and Warrington, 1990).

The limitations of short-term memory have an enormous affect on interface design, as it places a limit on the number of artifacts and tasks the user can keep track of. Errors can occur when these limits are exceeded (Norman, 1990).

2.2.2 Long-Term Memory

Long-term memory is responsible for storing a myriad of information, including motor skills (such as how to walk or touch-type), problem-solving skills (solving the Towers of Hanoi problem, for example), language skills (associating names to objects), and memories of past events. It is this last type of memory, 'episodic' or 'autobiographical' memory, that is the most relevant to the design of a temporal document-retrieval system, as creating and editing a document is an 'event'.

When an event is recalled, a datum, or 'cue', is used to trigger the recall. Experiments into recall typically start by providing participants with pairs of items to memorise. Later, the participants are asked to recall one part of the pair with the other given as a cue. These experiments have concluded that

recall is less accurate when the pair is random, such as 'white, 778', compared to when the pair has a relationship, such as 'white, black' (McCarthy and Warrington, 1990). Interestingly, people are no more likely to remember 'bizarre' events compared to common-place events even though people *thought* they were more accurate at recall (Winograd and Soloway, 1986; Worthen and Wood, 2001). In addition, people find it difficult to associate bizarre events with objects (Worthen and Wood, 2001). However, recall is more accurate when more information is given as a cue to recall (McCarthy and Warrington, 1990).

It appears that "dates usually are not explicitly represented in memory", even though autobiographical memory is a chronology (Larsen et al., 1999). Research into using time as a cue shows that people are not good at remembering *when* an even occurred, but are good at remembering the *order* of events (Michon, 1972; Loftus and Marburger, 1983; Wagenaar, 1986; Larsen et al., 1999; Anderson, 2005). Czerwinski and Horvitz (2002) found that people over-estimate time *periods* by 144.9%, while Larsen et al. (1999) showed that people are able to remember the exact *date* of an event 10% of the time.

2.2.3 Recall from Memory and HCI

Research into human memory suggests a number of guidelines for HCI that are related to temporal retrieval.

- As the size of short-term memory is limited, the number of interface artifacts that the user must keep track of should be limited, or the interface will become hard to use.

- An interface should supply as many cues to recall as possible, as this should aid retrieval of the sought item.

- Cues should not be bizarre, as people have difficulty remembering bizarre cues.

- Finally, as a cue, temporal order is more important than dates, as people are good at remembering temporal order, but poor at remembering dates.

2.3 Tasks with Documents

In the creation of a temporal document-retrieval system, it is important to understand people's tasks with documents. According to Malone (1983), there are two tasks carried out with document retrieval-systems.

- If documents are used for what is now termed 'information foraging' (Pirolli and Card, 1999) then they are "explicitly titled and arranged in a systematic order" (Malone, 1983), to allow them to be *found* more easily.

- Documents can also be used to *remind* the user of his or her current tasks. Malone (1983) found that in this case documents are "not, in general, arranged in any particular order" except for a "haphazard" temporal order. This allows the user to quickly scan the haphazard 'piles' and see which documents were viewed, created, or moved most recently.

Malone (1983) concluded that systems needed to support *both* the finding and reminding tasks, as both were equally important.

In this section I look at the two document centric user-tasks: finding (Section 2.3.1) and reminding (Section 2.3.2). I then conclude by looking at the impact on the document retrieval tasks on HCI (Section 2.3.3). However, first I define 'information' and 'documents'.

I will define information as data that increases a person's knowledge. Data can be encoded in many ways — including hand gestures, song or written words — but I will confine myself to discussing digitised data that can be processed by an electronic computer.

What a person considers to be a document to be depends on the context. For example, a book may be considered a document when it is retrieved from the library, but an article in the book is considered a document when the person is writing a review. I will assume that what the user considers a document is reflected accurately by the computing system. If this is not the case the usability of the system would be reduced because the user's mental model of the system would not reflect the system image (Figure 2.3).

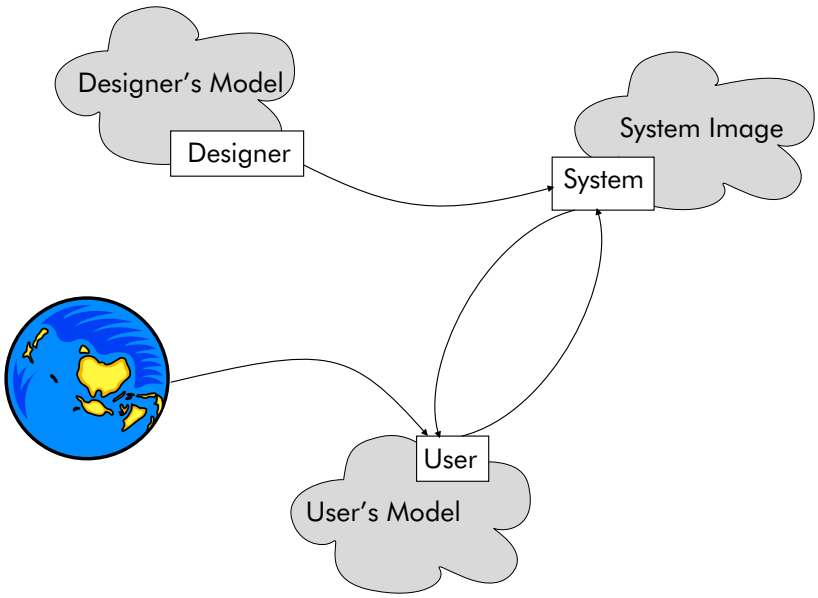


Figure 2.3: The model of interaction by Norman (1990). The usability of the system will be reduced if the user's model, system image and designer's model do not agree.

2.3.1 Finding

Finding information is an important user task as "humans actively seek, gather, share, and consume information to a degree unapproached by other organisms"(Pirolli and Card, 1999). Users' behaviour when retrieving information can be modelled by information foraging, which is based on a biological food-foraging model (Pirolli and Card, 1999). Information is collected in 'information patches', which are analogous to berry patches. Foraging starts in a patch that the user thinks is likely to be profitable. The user then forages through patches, seeking information and assessing if the search should continue in the current patch or move to another patch. The task of locating information is nearly continual, as the user is seen as an 'infomavore' whose main task is the location of information.

It appears that a user's behaviour is different when retrieving a document that he or she authored. Barreau and Nardi (1995) discovered that typical users only keep a few documents on their virtual desktop, and these documents were retrieved frequently. However, once finished the documents were

filed and were rarely retrieved: in their study they "found that users do not expend great energy on archiving because old information is generally not useful information." Beyond the work of Barreau and Nardi (1995), there has been little research into how people retrieve their own documents that I am aware of.

Information foraging cannot be used to model a user's behaviour when his or her own documents, which have been written recently, are being retrieved. Such documents do not increase the user's knowledge because he or she knows nearly all the information contained in the document. To borrow a term from information theorists, the 'entropy' of a document for the author is close to zero. Technically, the information-foraging model uses the following equation to determine if a user will stay in the current patch, or move to another patch.

$$R = \frac{G}{T_B + T_W}$$

It states that the rate-gain of information in a patch (R) is equal to the net information gain of the patch (G) divided by the time spent between patches (T_B) and the time spent within the patch (T_W). The user selects the patch with the greatest R value and starts foraging. For a user's own documents, G would be nearly zero, as the user knows almost all the information contained within the document, so the resulting R value would be close to zero. If the user forgot what was in the document the above argument does not hold, but for recently written documents the above formula means a user would never retrieve his or her own documents under the information foraging model.

While it is unfortunate that information foraging cannot be used to predict users' behaviour when searching documents, I will assume that the units of information, or 'cues' (Section 3.1), that can be used to locate documents is the same as the cues that can be used for information foraging.

2.3.2 Reminding

Lamming et al. (1994) report that 33% of recall problems involve 'prospective memory': the memory of tasks to undertake in the future (Lamming et al., 1994; Czerwinski and Horvitz, 2002).

One of the ways remininding is supported in the physical world is by cre-

ating 'piles' of documents (Malone, 1983). These piles are in a "haphazard" temporal order, with the most recently viewed documents at the top of the pile.

Electronic documents can serve a similar purpose: Barreau and Nardi (1995) found that users typically kept documents related to current tasks on the virtual desktop. Once the task was complete, the related documents were moved from the desktop to reduce clutter.

2.3.3 Document Tasks and HCI

The findings of Malone (1983) suggest that computer-based document retrieval systems must support two tasks — finding and reminding — that do not appear to be related. Barreau and Nardi (1995) also found that a user typically will not retrieve an old document that he or she has written. However, the documents that the author *does* write and retrieve also act as reminders for current tasks. Therefore I will further refine the tasks that *personal* document organisation systems have to support to

- Find recently viewed documents, and
- Remind the user of the current tasks.

Documents will also need to be stored, which may be a separate task depending on the document organisation system.

The findings also suggest that users' behaviour when retrieving their own documents for editing is different than when retrieving documents for information gain — which is modelled by information foraging (Pirolli and Card, 1999). Systems will, therefore, have to support two different finding tasks: finding the author's own documents, and finding documents for information gain.

2.4 Version-Retrieval Tasks

The retrieval of a past version of a document can be seen as a specific form of document retrieval. The ability to retrieve different versions of a document

allows the user to correct errors and explore system functionality (Shneiderman, 1997; Cooper and Reimann, 2003). In this section I examine errors in general, how errors can be corrected by retrieving prior versions of a document, potential cognitive issues with recovering from errors by retrieving a prior version of a document, and how system exploration is supported by version retrieval.

There are two types of user-error (Norman, 1990):

Slips are errors that occur *without* conscious thought, such as pressing adjacent keys (the 4 key instead of the 5 key) when typing;

Mistakes are errors that involve conscious thought, such as misinterpreting a dialog box.

I am not concerned with *why* errors occur — which is discussed by Norman (1990) — but how a person can recover from an error made when creating an electronic document.

There are two ways a person may recover from an error that has been made in an electronic document. First, the user can use 'forward error-correction' (Abowd and Dix, 1995): creation and deletion actions are used to remove the error. In the physical world forward error-correction is generally the only mechanism for correction of errors, a situation that is caricatured in Figure 2.4, but in the electronic realm forward error-correction may be augmented with the ability to retrieve a version of a document where the error does not exist.

The different mechanisms for retrieving a version of a document are examined in Section 3.3. For the remainder of this section I will examine the different error-recovery subtasks that can be completed with version retrieval, and the potential cognitive difficulties associated with this.

I will mainly consider two error-recovery subtasks: linear- and branched error-recovery (Figure 2.5). To illustrate these two tasks, consider a document that is being edited. As actions are completed, the version of the document moves from Version A to Version B and onto Version C. 'Linear recovery' is when the current version (C) is abandoned to return to the prior

Figure 2.4: The absence of version retrieval, specifically undo (Section 3.3.1.1), is lamented in this comic by Parisi (1991).

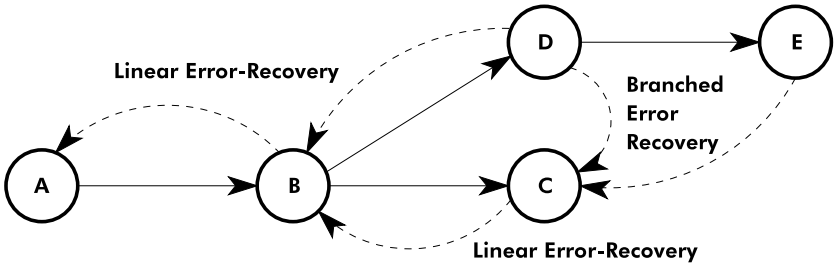


Figure 2.5: The versions created in an editing task. State A is the initial state that is edited to create State B. This is edited to create State C, but this editing is abandoned (linear error recovery) to return to State B. State B is edited again to create State D. Finally, this state is edited to create State E. Branched error-recovery is returning to State C from D or E.

version (B). If editing is continued from Version B, a new version (D) is created. 'Branched recovery' — which is also known as 'change of heart' (Vitter, 1984a) — is when the user returns to Version C from he erroneous Version D.

To perform either linear or branched error-recovery the user must realise that an error has been made *as soon as it occurs* or work will be lost. For example, consider the case where Version E is created after D: the version-retrieval system cannot be used alone to revert to Version C (branched error-recovery) without losing what was created in Version E.

Correcting an error by retrieving a prior version of a document may be cognitively difficult. Dix et al. (1997) state that when a document is reverted to a prior version "you are saying 'my last action was wrong' that is, you are thinking *about* what you are doing, instead of doing it", which is a breakdown situation. They argue that such a breakdown situation would hamper the usability of systems that use the retrieval of prior document versions to correct errors. A study that partly attempts to determine if the claim by Dix et al. (1997) is correct is presented in Chapter 5.

Shneiderman (1997) states that the ability to retrieve a prior version of a document encourages the user to "explore unfamiliar options" as the user can try a feature, to determine what it does, without having to know what actions are required to revert the document using forward error-correction. Such exploratory actions — even if a document is modified to create an undesirable version — can not be constituted a mistake (an error in conscious thought) as the user's task is to explore a feature of the system. However, if the document needs to be reverted to a prior state then the error-recovery subtasks outlined above are able to be used to retrieve a prior version of the document.

2.4.1 Version Retrieval and HCI

There are three main factors of version retrieval that affect HCI.

1. Forward error-correction is the primary mechanism to correct errors in the physical world, rather than version retrieval. Because of this, there is no common metaphor that can be used to make version retrieval easy to learn.

2. However, version retrieval should be a useful way to correct errors as the user does not have to formulate forward error-correction actions to recover from an error. The ability to revert to a prior document state without having to carry out forward error-correction also allows version retrieval to support system exploration (Shneiderman, 1997; Cooper and Reimann, 2003), which should make the system as a whole easier to learn.

3. Finally, the retrieval of a prior version of a document may cause a breakdown situation. It is also unknown whether it is possible to alleviate the potential breakdown situation caused by retrieving a prior version of a document.

2.5 Conclusion

In this chapter I looked at four human factors that affect temporal document-retrieval: perception of time, human memory, people's tasks with documents, and people's tasks with document versions. A number of guidelines were extracted from the existing research into the human factors (Sections 2.1.4, 2.2.3, 2.3.3 and 2.4.1). The guidelines are used to inform the study of systems in Chapter 3, the evaluations in Chapters 4 and 5, and the design of a system that combines temporal document retrieval and version retrieval (Chapter 6).

Chapter III

The Organisation of Documents and Versions: Systems and Approaches

In this chapter I will provide insight into how existing systems support the user's tasks with documents and versions. By looking at existing systems, and identifying features and flaws, I will create HCI guidelines for systems that use time to organise documents and versions, and identify research opportunities.

I start by examining cues, what they are and how they are presented by various systems. Like the research presented in Section 2.1, much of the research presented in Section 3.1 supports all three of my research themes: the retrieval of documents, the retrieval of versions, and the combined retrieval of documents and versions. For the most part, the systems described in this section are non-temporal document organisation systems, but the study is used to inform the discussion of the temporal systems presented in the next two sections and Chapter 6.

In Section 3.2 I examine how cues are used by temporal document-organisation systems to support three of the user's tasks: finding, reminding and storing. This section relates to the first of my research themes — using time as a method of organising documents — so it relates to the other sections that belong to this theme: Section 2.3 and Chapters 4 and 6.

Version retrieval, the second of my research themes, is covered in Section 3.3, where I examine how cues are used in version retrieval. This study leads on from Section 2.4, and the results are used in Chapters 5 and 6.

Finally, the conclusion of the chapter is presented in Section 3.4.

3.1 Cues to Document Retrieval

Cues were briefly introduced in Section 2.2.2, in the context of recall from long-term memory. In this section I will examine cues in more detail, looking at cues in the context of systems.

I start by examining what a cue is, and discuss how the possible number of retrieval cues is endless (Section 3.1.1). The specific set of cues that will be used for classifying systems, and how they are supported by different non-temporal document organisation systems, will then be discussed (Section 3.1.2). I conclude this section with a discussion of some HCI guidelines and open research questions in Section 3.1.3.

3.1.1 Cue Defined

I define a cue to be data about or from a document. There are two possible ways that a cue can be used for retrieval. First, cues can be presented to the user, reminding him or her of the documents that are stored; this allows the user to match what is recalled about the document against the information presented. For example, an email client can present sender, title and date information, which can help the user find a message sent by a colleague yesterday. Second, the user can present some cues to the system, such as text from the body of a document, that is used by the system to retrieve documents that match those cues. Throughout this thesis I will concentrate on the former, as it is the most common method of using cues in temporal document-organisation systems (Section 3.2). However, I will make reference to the latter method, which is used in full-text retrieval systems and search engines (Blair and Maron, 1985; Cunningham and Connaway, 1996; Freeman, 1997; Brin and Page, 1998; Witten et al., 1999; Pitkow et al., 2002; Yahoo!, 2005).

A cue is part of the context of a document. Context is hard to define (Abowd and Mynatt, 2000; Coschurba et al., 2001; Dey, 2001; Erickson, 2002; Powers, 2003) but Dey (2001) provides a useful definition:

> Context is any information that can be used to characterise the situation of an entity. An entity is a person, place, or object that is con-

sidered relevant to the interaction between a user and an application, including the user and the applications themselves.

This definition of context is broad, but necessarily so as a user's context is vast (Erickson, 2002). For example, the context for viewing or editing a document may include what the weather was like, what the user was drinking, who else was in the room, what the others were wearing, what others were doing, what the user had for dinner the previous night, what the user has planned for the weekend, what phone calls and mail messages the user received, what music was playing...

A large amount of 'ubiquitous computing' research has investigated how to capture context. Abowd and Mynatt (2000) and Bardram (2004) point out that most of this research has been into ways to determine *where* the users are, with systems such as Pepys (Newman et al., 1991; Lamming et al., 1994) and the Active Badge (Harter and Hopper, 1994; Harter et al., 2002; Schilt et al., 2002). While the research into ubiquitous computing is informative to my work on document retrieval, it tends to ignore the more traditional forms of context, which are discussed in Section 3.1.2.

3.1.2 Document-Retrieval Cues

The list of possible retrieval cues is endless, as every part of users' life-experience could be used as a cue to document retrieval (Section 3.1.1). However, there are some standard sets of retrieval cues. One such set is defined by the Dublin Core Metadata Initiative: it lists fifteen 'elements' that can be used as cues to document retrieval (Dcmi Usage Board, 2003b; ISO, 2003). It was developed as part of a wider effort to attach 'semantic' information to documents on the World Wide Web (W3C, 2005). Currently, the Dublin Core is used in over 73 projects including The European Libraries and Electronic Resources in Mathematical Sciences, The National Library of the Netherlands, and Te Kete Ipurangi (Dcmi Usage Board, 2003a). Throughout this thesis I will use the cues defined in the Dublin Core to classify systems and interfaces.

In this section I will discuss the elements defined by the Dublin Core Metadata Initiative and show how they are used by some example systems,

	Office Files	File Managers	PhotoMesa	Flatland	Data Mountain	ACM Digital Library	Google
Title	★	★			✔	★	★
Creator	✔	✔				✔	
Subject						✔	✔
Description	✔		★	★	★	✔	✔
Date		✔				✔	
Format		★				✔	
ID		✔					✔

Table 3.1: Common retrieval cues and their support in various document-retrieval systems. A ✔ indicates that a cue is supported by the system, while a ★ indicates that the cue is an important one for retrieval using a particular system.

but I will not directly discuss the cues used for Web page retrieval. In principal, any XHTML document on the Web can be linked with a companion Resource Description Framework (RDF) file (Powers, 2003; Beckett, 2004). This file can contain a limitless number of cues, such as Dublin Core metadata (Beckett, 2002), a description of the author and *all* the people that he or she knows (Brickley and Miller, 2005), and licencing rights for the document (Creative Commons, 2005). While I am aware of the research into the 'Semantic Web', for the most part it is outside the scope of this thesis and it will provide little insight into temporal document-retrieval.

Seven common cues will be used to classify systems: title, creator, subject, description, date, format and ID. Table 3.1 shows which of these cues are supported by some example systems. The systems were chosen because they are commonly used — such as office files (Malone, 1983), file managers and Google (Brin and Page, 1998) — or clearly illustrate a method of document retrieval. (Temporal document-organisation systems and version-retrieval systems, which are not listed in Table 3.1, are covered in Sections 3.2 and 3.3.)

Title The name of the document. File managers require the names of all documents contained in a particular folder to be unique (Corbató et al., 1962; Daley and Newman, 1965; ver Hoef, 1966; IBM, 1982; Takatsuka et al., 1986). However some systems, such as email clients (Section 3.2), allow multiple documents to have the same title. Other cues, such as dates or ID, are used to distinguish documents in such cases. In some systems the length of the title may be limited, such as file-names on compact disks that are limited to 30 characters (ECMA, 1987). Interestingly, many of the systems listed in Table 3.1 that provide a description of the document as an important cue do not provide titles as a cue.

Creator The person (or generic entity) that created the document. On multi-user systems, file managers automatically attach creator information to the document. In addition, documents are often organised by creator: the 'home directories' on Unix systems (Russell et al., 2004) and the 'userprofiles' under Windows (Karp et al., 2002).

Subject Keywords that describe the document. Keywords can be automatically extracted from documents (Jones and Paynter, 2001) or manually generated and associated with documents, as is done with the 'category' of some pages returned by Google, as can be seen in Figure 3.1.

Description "An account of the content of the resource" (DCMI Usage Board, 2003b). What forms the description can differ from system to system. In Google the description is text extracted from the document. The Data Mountain Web-page bookmark organiser (Robertson et al., 1998) provides a description on the form of a thumbnail, which may contain a mixture of text and images. Many image organisers, incuding PhotoMesa (Bederson, 2001), also provide small thumbnails of images as a retrieval cue.

Date The date the document was created, viewed or modified. After being automatically assigned to documents, dates can be presented to the

Figure 3.1: A single result from a search for 'eiffel' in Google, showing eight cues to retrieval. **1.** The page title. **2.** The colour of the link (blue) indicates that the page has not been visited by the user in the past. **3.** A link reading 'Translate this page' that indicates that the page is not in English. **4.** The format of the page is HTML unless indicated other otherwise. **5.** A sample of the page with words that match the search-term emboldened, which provides a description of the page. **6.** A classification of the page. **7.** The size of the page. **8.** The ID of the page. The order of the pages returned by Google forms the final cue: relation.

user, as is often done with file managers.

Format The type of document. File managers typically use an icon to represent the format of a document, while Google textually indicates the type of document that is linked.

ID A unique identifier for the document. For file managers each document can be identified by a combination of the document title and the folder hierarchy, or 'path', that contains the document. Similarly, documents returned by Google have a unique Uniform Resource Indicator (URI: Berners-Lee et al., 1998).

There are eight elements described by the Dublin Core that are not listed in Table 3.1. Seven are relatively unimportant, and are described below; the relation cue is almost universally important, and is described in Section 3.1.2.1.

Publisher Who made the document available. Often reviews of books or audio-recordings will mention the publisher of a document, but none of the systems mentioned in Table 3.1 provide the publisher as a cue to document retrieval.

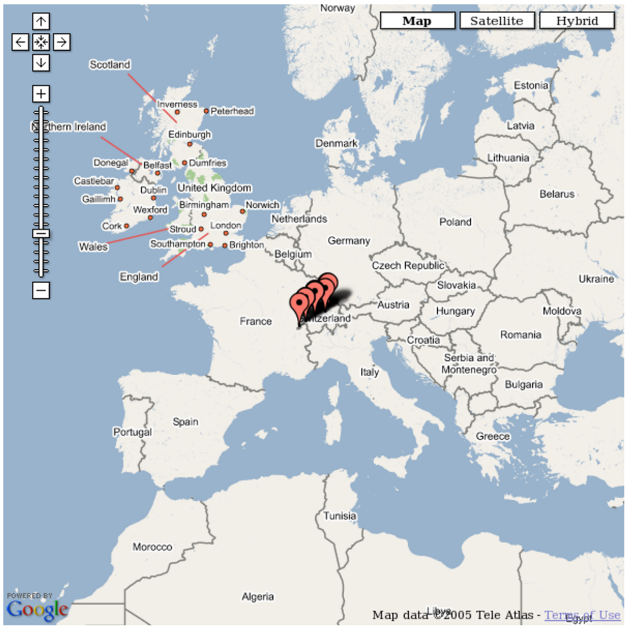


Figure 3.2: A map of 'podcasts' — regular audio-shows provided over the Internet — from Switzerland, as displayed on Google Maps (PodcastDirectory.com, 2005; Google, 2005).

Type "The nature or genre of the content of the resource" (DCMI Usage Board, 2003b). Music-files are often associated with a particular genre (Nilsson, 1999; Xiph, 2004). A problem with associating a music file with a genre is that it may belong to multiple categories, such as baroque and classical. To my knowledge, it is not common practise to associate documents with genres in document-retrieval systems outside the music domain.

Coverage The geographical area described by the document. For example, the Ricoh Capilio Pro G3 digital camera (Ricoh, 2005) can record the longitude and latitude of the camera when a photograph is taken, and this can be used to organise the documents. Google Maps (Google, 2005) extracts coverage information from documents and allows the user to search for documents that discuss a particular location, such as all Internet audio-shows broadcast in Switzerland (Figure 3.2). Coverage is distinct from other geo-spatial information such as the address of the

publisher or conference location: documents are associated with these locations but do not necessarily *cover* these locations. Coverage information may become more common with increased use of the Global Positioning System (GPS), but it is uncommon compared to the cues listed in Table 3.1.

Contributor Additional authors. Most systems either only allow one author to be used as a cue to retrieval (as is the case with file systems), or combine all authors into the same cue, as is done with the ACM Digital Library (Association for Computing Machinery, 2004).

Rights The copyright and licensing terms for the document. I am only aware of the Yahoo! Advanced Search (Yahoo!, 2005), and its precursor the Creative Commons Search Engine (Creative Commons, 2004), that allow the retrieval of documents according to the licensing restrictions.

Source The document from which the current document is derived. For example, The ACM Digital Library (Association for Computing Machinery, 2004) lists the conference proceedings, or journal, that an article was originally published in. Citations also often include the source document of the article, such as those for Abowd and Dix (1995), Abrams et al. (1998) and Abu-Shakra and Fisher (1998). Of the systems listed in Table 3.1, only The ACM Digital Library provide source information as a cue.

Language The language the document is written in. Most systems do not specify the language the document is written in, but Google determines the language a Web page is written in by analysing the content of the document (Figure 3.1) .

There are two notable features of Table 3.1. First, the full-text retrieval systems The ACM Digital Library and Google supports all of the cues in Table 3.1 between them. (The ACM Digital Library also supports the source cue, which is not listed.) Second, many systems only support two cues.

The Google search engine illustrates how many cues can be presented in textual form. Figure 3.1 shows a single result from a search using Google.

Each result contains six cues that are present in the Dublin Core: title, language, format, description, subject (category), and ID. The position of the result in the page — that is, its relation with other documents (Section 3.1.2.1) — forms the final cue. There are two cues presented by Google that are not covered by the Dublin Core: the colour of the link indicates that the page has been visited by the user in the past, and the size of the page is also listed. The date returned by Google does not indicate the age of the page, but the date an *experimental* 'crawler' visited the page gathering data, rather than the date that the main search-engine indexed the page; the date is not part of the context of the document, but it does show that the page did exist on a particular day.

Between them, Google and the ACM Digital Library support all the document-retrieval cues listed in Table 3.1. However, many interfaces only allow few cues to be used to retrieve documents: the Flatland electronic whiteboard, the PhotoMesa image organiser and the Data Mountain Web-page bookmark organiser provide two cues each. The usability of systems that provide few cues *should* be lower than those that provide many cues according to findings from psychology: research indicates recall is easier when more cues are provided (Section 2.2.2).

3.1.2.1 The Relation Cue

The relation cue defines how the system organises documents. In many systems, the relation cue is formed by organising documents according to other cues — such as alphabetically sorting documents according to their titles — so it can be thought of as a meta-level cue. The relation is an important cue because it defines what information the user must be able to recall in order to locate a document quickly and easily. For example, the relation cue is important for temporal document-organisation systems (Section 3.2), because locating the document may be more difficult if the user cannot remember the temporal relationship a document has with the other documents in the system.

Krishnan and Jones (2005) list five ways to organise documents.

1. In a 'hierarchal' system documents are placed in a classification hier-

archy, such as the files and folders employed by many systems.

2. Documents organised in a 'network' are linked together in a mesh, such as hypertext systems (Bush, 1945; Nelson, 1965; W3C, 1999).

3. Examples of systems where documents are organised in a 'spatial' way include the the standard desktop, Data Mountain, and the Flatland electronic whiteboard (Edwards et al., 2000; Mynatt et al., 2000).

4. 'Activity based' systems group documents according to particular activities, such as email and Web browsing. Examples of activity based organisers include Rooms (Card and Austin Henderson, 1987), TimeSpace (Krishnan and Jones, 2005) and the 'virtual desktops' found on many Linux and Unix systems.

5. Finally 'temporal' systems order documents according to a temporal property (Section 3.2).

I will add a sixth organisation method to this list: 'relevancy'. With this method, the relationship cue is based on the document's relevancy to a cue supplied by the user, as is done by full-text retrieval systems, such as Google.

3.1.3 Cues to Document Retrieval: Conclusion

The following are some guidelines and recommendations that can be drawn from prior work into document retrieval cues.

- When designing a document retrieval system, the context of the user and the documents must be determined, as this will assist in deciding which cues should be provided. The Dublin Core set of cues may be used as a starting point in deciding which cues to support.

- The relationships between documents is an important cue for document retrieval for almost all interfaces. Systems should make the relationship between documents clear.

- Systems should provide many cues, as research suggests that retrieval is easier when more cues are provided (Section 2.2.2). However, so the user is not burdened with creating cues, such as document titles, systems should provide cues that can be automatically determined — such as creator, date, description, and format, which are provided by many systems (Table 3.1).

Some open research questions are raised from this investigation. When should a system to automatically create inter-document relationships? The research into temporal document-organisation (Section 3.2) suggests that it is usable despite the few cues provided to the user, so when is it appropriate to provide few cues? Should cues be given in a textual form — such as that used by Google and the ACM Digital Library — or in a graphical form? Finally, Table 3.1 shows some cues that are supported by different systems, but *what* cues should be supported to make document retrieval easier for users?

My research concentrates on temporal cues — such as date and visit-order — because the pervasiveness of time, the reduced effort in forming inter-document relationships, and the users' ease at remembering the order of events make time a compelling method to organise documents and an effective cue for document retrieval. In Section 3.2 those benefits are discussed in greater detail, with a unique benefit, version control, discussed in Section 3.3.

3.2 Temporal Document-Organisation

In this section I look at temporal document-organisation systems, the benefits of organising documents temporally, and the drawbacks. I begin by looking at the cues that are provided by various temporal document-organisation systems (Section 3.2.1). Next, I look at how temporal document-organisation systems support the finding and reminding tasks (Sections 3.2.2 and 3.2.3). Then, in Section 3.2.4, the advantages of using temporal systems to store documents are examined. Finally, I discuss some example temporal document-organisation systems in Section 3.2.5, before presenting some HCI guidelines in Section 3.2.6.

	Office Piles	Email Clients	AutoAlbum	Back	Browser Histories	Lifestreams	Outride	PadPrints	Session Highlights
Title	✔	★		✔	★	★	★	✔	✔
Creator	✔	★							
Subject									
Description	★		★			✔		★	★
Date	✔	✔		✔	✔				
Format					✔				
ID									✔

Table 3.2: The support of common retrieval cues in temporal document-organisation systems. A ✔ indicates a cue that is supported by the system, while a ★ indicates a cue that is important for retrieval using a particular system.

3.2.1 Cues in Temporal Document-Organisation

Table 3.2 shows the cues (from Section 3.1) that are supported by some temporal document-organisations systems. Table 3.2 is sparse, with systems supporting few cues compared to the non-temporal document-organisation systems (Table 3.1).

The defining characteristic of temporal document-organisation systems is the primary relationship between documents: the relation cue, which is formed by ordering documents based on a date. The particular date used can differ between systems: email clients — such as Microsoft Outlook, Novell Evolution (Moulder et al., 2004) and pine (University of Washington, 2002) — use the date a message was received as the basis for organising messages; the AutoAlbum image organiser (Graham et al., 2002) uses the creation date for the photographs; the time the document was last edited or viewed forms the basis for document organisation in the Lifestreams system (Freeman, 1997) and the remainder of the systems listed in Table 3.2. In the case of office piles the temporal ordering is "haphazard" rather than a strict ordering (Malone, 1983). The other cues supported by the systems listed in Table 3.2

are similar to those in non-temporal systems.

Some systems listed in Table 3.2 are able to organise documents in a non-temporal manner — such as employing folders in an email client — while some of the systems listed in Table 3.1 are able to employ temporal document-organisation, such as ordering files by modification date in a file manager. In such cases, I have listed the systems in the particular sections based on what I consider to be the primary relation cue employed by the system.

In Section 3.2.5 I will look at the specifics of two document retrieval systems: the back button in Web browsers, and history lists. However, first I will examine how temporal document-retrieval systems support the user's tasks with documents: finding, reminding, and storing (Sections 3.2.2–3.2.4).

3.2.2 Finding Documents

In this section I will examine at how well temporal document-organisation systems support the finding task, which was first examined in Section 2.3.1. I begin by looking at the results of others' research that suggests that some temporal document-retrieval systems are used frequently and allow for fast retrieval of documents. I then conclude by looking at two disadvantages of temporal document-retrieval: the users' difficulty in remembering dates, and the retrieval of old documents.

There is some evidence that users find temporal document-organisation systems useful. For example, the back button in Web browsers accounts for 40.6% of user-interaction with Web browsers, second only to hyperlinks, which accounts for 50.9% of actions (Catledge and Pitkow, 1995).

There is evidence that temporal document-organisation is usable, as well as useful. In their study, Pitkow et al. (2002) found that users are faster at locating pages when using the Outride system, which uses visit-histories to reorder the results from search engines, than using search engines alone. In another study, Platt et al. (2002) found that users were not significantly faster at using the temporal PhotoTOC image-retrieval interface than any of the other interfaces. However, users did significantly prefer PhotoTOC to the other interfaces. The IM^3 system implicitly stores all documents that

have been photocopied, printed or faxed by the user (Hull et al., 1999). Hull and Hart (2001) found that users "could more easily retrieve a desired print document from the IM^3 database than from their own file directory."

One potential disadvantage of temporal document-retrieval is that people are not good at remembering the date of an event, such as viewing a document (Section 2.2.2). The theory that dates do not provide a good cue to document retrieval is supported in practise: neither the back button, AutoAlbum or Outride provide dates as a cue to document retrieval (Table 3.2), but all appear to be usable according to the respective studies (Cockburn et al., 2002; Graham et al., 2002; Pitkow et al., 2002). Instead of dates, these systems rely on temporal order, which people have a good memory for, to organise documents and cue recall.

A disadvantage of temporal document retrieval is that documents that were not seen recently would be harder to locate, as the user can potentially have a large number of items to look through. As discussed in Section 2.3.1, old documents are not frequently retrieved by their authors, so this should not be a significant problem when retrieving documents for editing.

3.2.3 Reminding the User of Current Tasks

According to Malone (1983), the reminding task is equally important as the finding task (Section 2.3). Temporal document-retrieval systems should support reminding exceptionally well: documents associated with current tasks will usually be at the either the start or end of the list (depending on how the list is sorted). Completed documents will move out of the users view without any explicit user action as newer tasks are added to the list. Non-temporal document-organisation systems cannot support reminding as easily, as users have to expend effort in organising documents when they are no longer needed to act as reminders, as Barreau and Nardi (1995) discovered (Section 2.3.1).

3.2.4 Storing Documents

Unlike many other systems, temporal document-organisation does not require the user to expend any effort in forming the relation cue, as temporal context

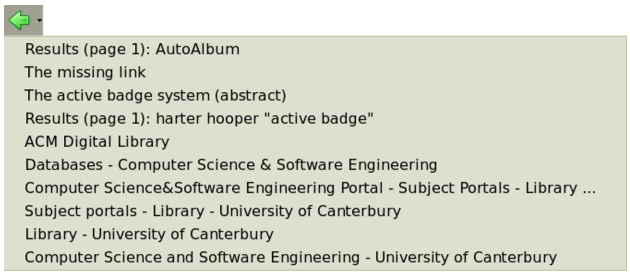


Figure 3.3: The back button in Mozilla Firefox showing page titles in the back menu. It is similar to the back button found in the other commonly used Web browsers: Microsoft Internet Explorer, Opera, Safari and Netscape Navigator (Aula et al., 2005).

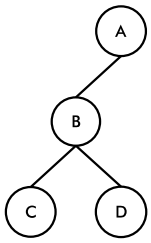
is used to automatically form relationship between documents. For example, email clients automatically organise messages by the date that the message was received, and history lists organise pages in the order they were viewed. This contrasts with other systems, where the user often has to explicitly form relationships between documents. For example, a classification of the document has to be created when office files or file managers are used to store documents.

3.2.5 Two Temporal Document-Organisation Systems: The Back Button and History Lists

In this section I discuss the specifics of how time is used to organise documents in two temporal document-organisation interfaces, the research into the systems that implement those interfaces, their benefits and problems. First, the back button is presented (Section 3.2.5.1). Then, in Section 3.2.5.2, I discuss the use of history lists. These systems were chosen because they are heavily used, in the case of the back button, or they form the basis of many other types of temporal document-organisation interface, in the case of the history list.

3.2.5.1 The Back Button

The back button is a mechanism that allows a user to return to Web pages that he or she has seen in the past (Figure 3.3). It organises documents



Action	Current Page	Back Stack	Forward Stack	History List
—	A	$	$	A$
Goto B	B	A$	$	BA$
Goto C	C	BA$	$	CBA$
Back	B	A$	C$	BCA$
Goto D	D	BA$	$	DBCA$

Figure 3.4: How the back button loses documents (Document C in this case) during the hub-and-spoke browsing of the example document-hierarchy shown on the left. The list of pages stored by the system proposed by Tauscher and Greenberg (1997), which does not lose pages, is shown in the right-hand column.

in a temporal order, but dates are not shown; the absence of dates should not affect the usability of the system significantly, as people have difficulty remembering dates (Section 2.2.2). Titles of documents can be used as a cue if the user displays the back-menu (Figure 3.3), which is equivalent to the Go-To menu found in earlier Web browsers. However, Cockburn et al. (2002) found the back menu was rarely used: the user typically relies on his or her memory of what pages have been seen previously in order to navigate back to a document. This is also backed up by Catledge and Pitkow (1995) who found that the use of the Go-To menu in XMosaic accounted for 2% of user actions.

The back button is a highly used temporal document-organisation system: 40.6% of all user interaction with a Web browser is with the back button (Catledge and Pitkow, 1995). This is despite the back button having sub-optimal utility, as it loses pages from its record of past visits (Figure 3.4). The loss of Web pages is especially noticeable when carrying out 'hub-and-spoke' browsing (Cockburn and Jones, 1996), such as viewing pages returned by a search engine: the results page (the hub) links to many other pages (the spokes). The user revisits the hub frequently, in order to visit more spokes, but the record of prior visits to other spokes is lost from the back button.

An alternate back button, which did not lose pages, was proposed by Tauscher and Greenberg (1997). Their 'temporal back-button' stored pages

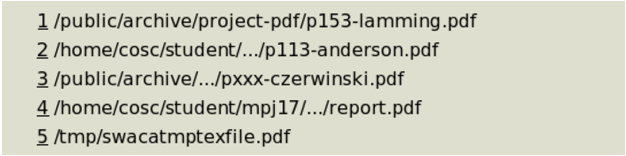


Figure 3.5: The file menu in Adobe Acrobat Reader, which allows the user to view portable document-format files, lists recently viewed files in the `File` menu, between the `Print` and `Exit` options. It is similar to simple history-lists found in many document editors and viewers.

in the order that the user visited them (with duplicates removed) rather than a stack. While the temporal back-button should allow users to retrieve pages faster than the standard back button, Cockburn et al. (2002) found that participants were no quicker at returning to Web pages compared to the standard back button.

Web browsers have introduced 'tabbed browsing', which allows the user to open the linked page in a new tab, rather than replacing the current page. Aula et al. (2005) report that experienced Web users employ tabs while searching (a type of hub-and-spoke browsing) so they do not lose pages. As browsers that support tabbed browsing become more common, the use of the back button may drop from the levels reported by Catledge and Pitkow (1995).

3.2.5.2 History Lists

In this section I will discuss some history list systems. A history list is a list of documents where the temporal ordering of the documents forms the relation cue. In their simplest form, such as the list of recently viewed files in Adobe Acrobat Reader (Figure 3.5), history lists are a commonly implemented method of returning to documents the user has viewed or edited in the past. They are quite simple interfaces, supplying three cues: ID and title (in the form of a file-name), and a relation cue (created by the temporal ordering of the documents). In addition to the lists in individual applications, most desktop systems provide history-lists in the form of 'recent files' lists, which supplement file managers (Apple, 2001; Willcox, 2002; Microsoft Corporation, 2005). However, in this section I will examine the more complex

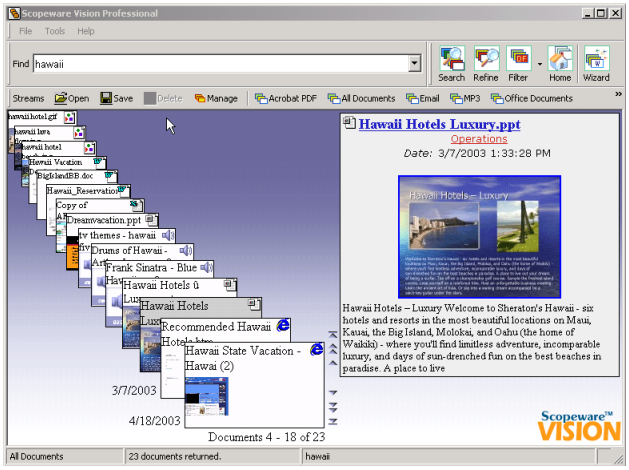


Figure 3.6: Scopeware (Mirror Worlds Technologies, 2003), the commercial variant of the Lifestreams system, showing a filter on 'hawaii'. Documents are arranged in a stack, with the most recently edited document at the front.

history lists that are found in the Lifestreams document organisation system (Freeman, 1997), Web browsers and digital image organisers.

The Lifestreams system is a replacement for the desktop that uses history-lists as the primary document retrieval mechanism (Figure 3.6). The ordering of the documents in the list (the relationship cue) is the same as that suggested by Tauscher and Greenberg (1997): ordered by the date the user last visited the document, and listing each document once. Unlike simple history lists, four retrieval cues are provided: title, description (in the form of a thumbnail), date, format, and relation. These are similar to the cues provided by a list-view in traditional file managers (Table 3.2). In addition, the user can create full-text searches (called 'streams') to reduce the number of items in the list and simplify retrieval.

There are three types of history lists that are found in common Web browsers (Table 3.3).

Linear lists Typified by Netscape Navigator 4 (Figure 3.7(a)), these simple lists present the pages in temporal order, with the most recently viewed page at the top or bottom of the list, depending how they are sorted.

Chunked lists Simple lists can get long, so chunked lists break the list into

Browser	Linear	Chunked	Category
Microsoft Internet Explorer			✔
Mozilla Firefox	✔	✔	✔
Opera	✔		
Apple Safari		✔	
Netscape Navigator ≤ 4	✔		

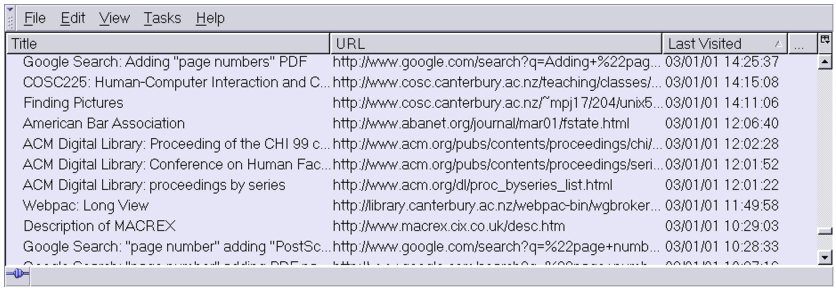
Table 3.3: The support for different types of history list in Web browsers: a ✔ indicates the type of list is provided. The browsers are those reported to be in common use by Aula et al. (2005), and Netscape Navigator, which was historically common.

days, or other temporal periods, to ease searching. Apple Safari 2 (Figure 3.7(b)) provides a chunked list in its `History` menu, showing all the pages viewed on the current day, and grouping pages viewed on different days in sub-menus.

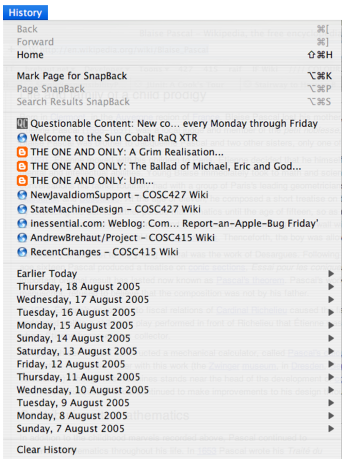
Categorised Trees A categorised tree breaks the history into non-temporal categories, as is done by Microsoft Internet Explorer 5 (Figure 3.7(c)). In Internet Explorer this hierarchy consists of temporal chunks (such as days or weeks) at the top level, alphabetically ordered websites at the second level, and alphabetically ordered pages at the third level.

Mozilla Firefox, the replacement for Netscape Navigator, supports all three forms of history list, with the user able to select which one is used.

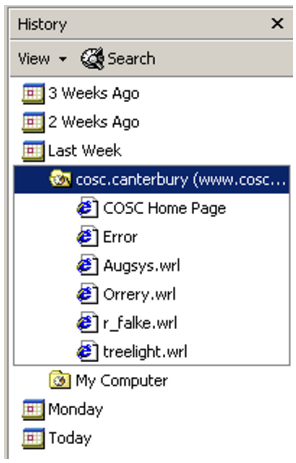
History lists in Web browsers should be useful: Cockburn and McKenzie (2001) found that for every new page that a person visits, four pages are revisited. However, Catledge and Pitkow (1995) reported that the use of history lists in Web browsers is low (0.1% of actions) compared to the use of hypertext links and the back button (50.9% and 40.6% of actions respectively). In their survey of self-selected 'experienced' users, Aula et al. (2005) reported that the browser history was not commonly used: the median response was that the history list was 'sometimes' used, as measured on a five-point scale (never, rarely, sometimes, often, almost always). I am not aware of any research that has discovered why history-lists in browsers are not more frequently used. Possibly it is because early forms of browser



(a) The History Window of Netscape Navigator 4, with the user's browsing history shown as linear list.



(b) The History Menu of Apple Safari 2 with the history broken into chunks.



(c) The History Pane of Microsoft Internet Explorer 5, with the history broken down by temporal chunk and site.

Figure 3.7: The history lists of Netscape Navigator 4, Apple Safari 2 and Microsoft Internet Explorer 5.

history-lists, such as those studied by Catledge and Pitkow (1995) in XMosaic, were cleared when the browser was closed. In addition, some history lists were implemented using a stack, which lost pages, rather than in a list that did not lose pages, such as that suggested by Tauscher and Greenberg (1997). Alternatively, history lists may be harder to access than the back button or links: they cannot be accessed through a single click, and may spend most of their time closed.

Some have speculated that the low use of the history lists in Web browsers may be due to low functionality, so alternate browser histories have been created. An example is PadPrints (Hightower et al., 1998). Rather than keep documents in a linear list, it organises pages according to the linking-structure, presenting the user with a hierarchy of pages (Figure 3.8). In addition to the hierarchy, which forms a relation cue, each page is shown with its title, and a thumbnail of the page (a description cue). The user could also control how much of the page's context is viewed by zooming: zooming out would reduce the detail of each thumbnail (description) but allow the user to see more of the hierarchy (relation). In an evaluation of PadPrints, Hightower et al. (1998) found that users were no faster at retrieving pages using PadPrints than back, but the participants did like the hierarchy.

Another system that aims to be more useful than standard Web histories is Session Highlights (Jhaveri and Räihä, 2005). Session Highlights differs from a standard browsing history in two ways:

- The user explicitly adds pages to the session history and
- Thumbnails are displayed, rather than page titles.

The page title and URI (an ID cue) is displayed when the user moves his or her mouse pointer over the thumbnail of the page, providing two more cues in addition to the thumbnail (description) and temporal order (relation). Session Highlights is similar to bookmarks (Abrams et al., 1998) as users have to explicitly add pages to the history. Unlike bookmarks, the pages are not stored in a user-defined hierarchy, but in temporal order. In their study of Session Highlights, Jhaveri and Räihä (2005) found that participants did

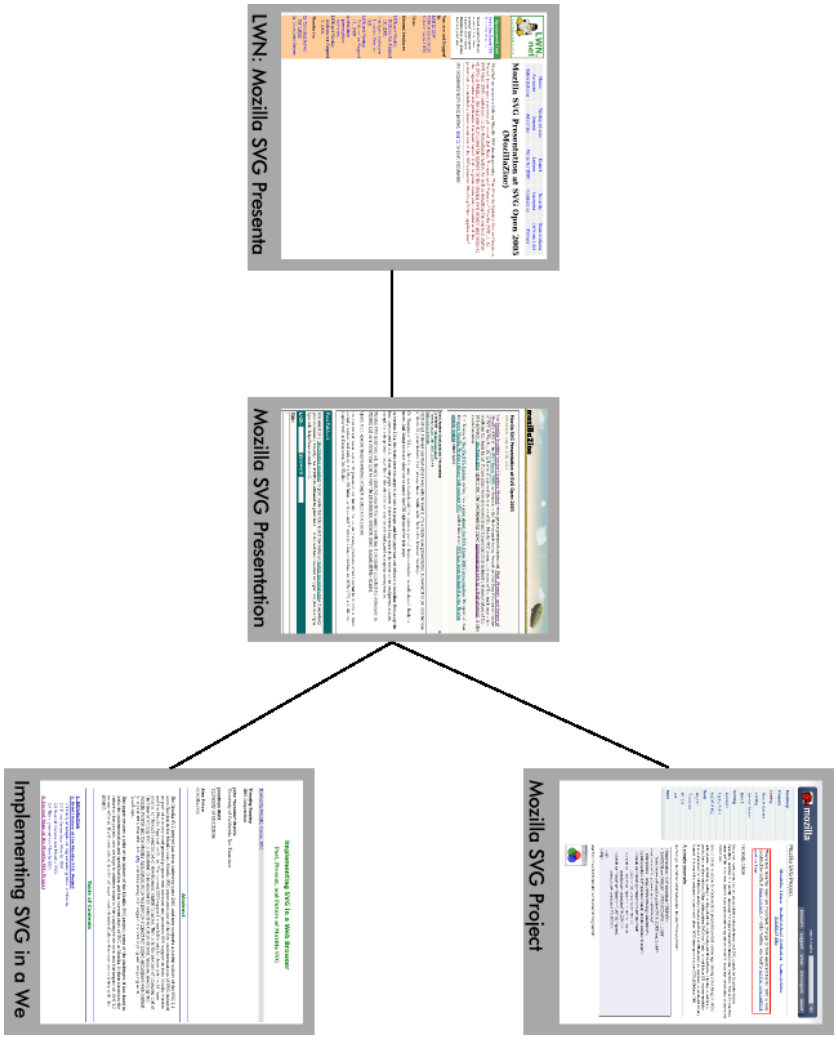


Figure 3.8: A mock-up of the PadPrints interface, based on Hightower et al. (1998). The browsing hierarchy is the same as that shown in Figure 3.4, with the middle page (Mozilla SVG Presentation) acting as the hub with two spokes coming off it. The pages are added from left to right, top to bottom: the bottom-right page is the most-recently added.

use the system during browsing tasks, but it is unknown if the system would be used outside the laboratory environment.

Digital cameras typically use time to order the large number of unnamed photos taken by users (Graham et al., 2002). The digital-photograph organisers PhotoTOC (Platt et al., 2002) and AutoAlbum (Graham et al., 2002) do not simply order images by time: an algorithm groups images, based on the observation that related photographs are often created together with large periods of time separating groups. In their study, Platt et al. (2002) found that users preferred the PhotoTOC system over the interface that did not create groups. It appears as if temporal-grouping schemes may be applicable to a wide range of document types: Abrams et al. (1998) noticed that related Web pages were often viewed during short periods of time, or 'episodes', that are similar to the groups in PhotoTOC. Lamming et al. (1994) also observed episodes in the location-data gathered by the Pepys system, which tracks where the user is and presents the information in a history list.

3.2.6 Temporal Document-Organisation: Conclusion

The following are some general guidelines that can be drawn from the existing research into temporal document-retrieval systems.

- To better support reminding, systems for organising the user's personal documents should make finding the recently viewed documents particularly easy, even if the retrieval of old documents is made more difficult.

- Dates can be used as a cue for document retrieval, but they are not necessary for temporal document-organisation interfaces. Many temporal document-organisation systems do not supply dates as a cue, such as back, and research suggests that they are still useful and usable.

- It appears that titles are an important cue to retrieval, as many interfaces provide document titles as a cue: office piles, email clients, back, history lists, Lifestreams and Outride.

- Creating temporal groups should make the retrieval of temporally organised documents easier.

Temporal document-retrieval is a compelling way to organise documents because it effectively reminds the user of the current tasks, it allows the user to retrieve recently viewed files quickly and easily, and documents are organised without user effort. However, the best way to present the information to the user is an open research question. Four possible ways to organise items in temporal document-retrieval systems will be investigated in an empirical evaluation in Chapter 4.

3.3 Version-Retrieval

Until now, I have only considered systems that store a singe version of each document: the most recent version. It is possible to store multiple versions of a document, so the user can retrieve the prior document state in order to correct errors or explore the system functionality — as was discussed in Section 2.4. Version retrieval can be seen as a specialised form of temporal document-retrieval where prior document versions are retrieved, rather than completely different documents.

Table 3.4 lists a number of example version-retrieval systems and the retrieval-cues that they support. Version-retrieval systems organise versions temporally, much the same way that documents were organised temporally by the systems discussed in Section 3.2. In fact, the relationship between versions is the only cue provided by undo and Historian (Abu-Shakra and Fisher, 1998).

In this section I will examine some version-retrieval systems, using the cues from Section 3.1. I will divide the systems into two groups. The first contains systems that implicitly create versions as the user performs actions, and is discussed in Section 3.3.1. These systems typically provide the user with very few cues to aid version retrieval, as can be seen in Table 3.4. This is in contrast to the many of the systems in the second group, which is examined in Section 3.3.2. These systems typically provide the user with many cues, but also require the user to explicitly create versions. This section concludes by drawing some guidelines and discussing future work (Section 3.3.3).

	Implicit			Explicit			
	Undo	Undo Menu	The GIMP	Historian	VMS	CVS	MediaWiki History
Title					★	★	
Creator					✔	✔	✔
Subject			✔				
Description		✔	✔			✔	✔
Date					✔	✔	✔
Format					✔		
ID					✔	✔	✔

Table 3.4: Retrieval cues for some example version-retrieval systems. A ✔ indicates that a cue is supported by the system, while a ★ indicates that the cue is an important one for retrieval using a particular system.

3.3.1 Implicit Version-Creation: Undo

Implicit version-creation systems, such as undo, implicitly create a new version of a document when the user performs an action that alters the document state. Undo was not present in the earliest document editors, such as QED (Deutsch and Lampson, 1967); by 1984 undo was still being described as a feature that "*may* become fairly common" (Archer et al., 1984) [emphasis added]. However, now undo is common: so much so that implementing undo is a 'pattern' in software engineering (Gamma et al., 1995).

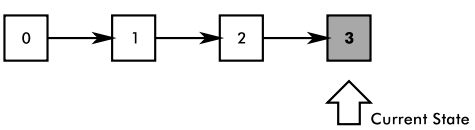
In this section I take a user-centred view of three different implicit version-creation systems. I begin by looking at the many different types of undo (Section 3.3.1.1), which only provide the relationship between versions as cue to document retrieval. This is in contrast to undo-visualisation systems (Section 3.3.1.2) that provide multiple cues to document retrieval, but are less common than standard undo. Finally, I discuss some other implicit version-creation systems in Section 3.3.1.3.

3.3.1.1 Undo

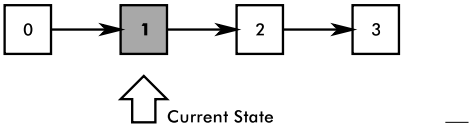
Using undo to retrieve the version of a document that was created immediately prior to the current version typically involves the user clicking a button labelled `Undo` or typing `Control-z`. It is present in most editing systems: from simple text editors such as vi and Microsoft Notepad, to complex systems such Adobe Photoshop and the applications in the Microsoft Office and OpenOffice.org suites. Undo can also be used to correct errors in systems other than text or graphics editors — such as operating-system shells (Holyer and Pehlivan, 2000), data-visualisation systems (Derthick and Roth, 2000) or electronic whiteboards (Mynatt et al., 2000). In most of these systems, the temporal ordering of versions is used as the principal retrieval cue: no visualisation is given as a cue to retrieval. (Systems that do provide a visualisation to undo are noted in Section 3.3.1.2.) In this section I will discuss three different forms of undo: bi-level undo, stack-based undo and history undo.

Bi-level undo, otherwise known single-step undo, records two versions of a document: the current version and the version immediately proceeding (Prakash and Knister, 1992). Activating undo discards the current version of the document and returns to the prior version; activating undo again would return to the version of the document before undo was first activated: "undo is its own inverse" (Joy, 1977). Bi-level undo was once the most common form of undo, found in systems such as ed and Bravo (Kernighan, 1979; Archer et al., 1984). However, due to the limited number of versions stored, the utility of bi-level undo is not as high as other forms of undo; interface designers have been encouraged to use stack-based undo instead (Shneiderman, 1997; Cooper and Reimann, 2003).

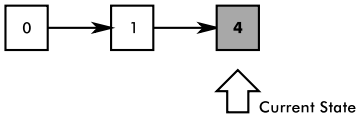
Stack-based undo (also known as linear undo or multi-level undo) stores document versions in a stack, with the top-most version being the current one. When undo is invoked, the current version is popped off the stack, allowing the user to access multiple versions of the document. The versions that are popped off the undo stack are added to the 'redo' stack; by invoking redo the user can return to a version that has been undone (Cooper and Reimann, 2003). Stack-based undo originated on the Cope environment



(a) The current version of the document, 3, is erroneous, but the user can fix the error by reverting to Version 1.



(b) Undo is invoked and the document is reverted to Version 1. Versions 2 and 3 can be reached by using Redo.



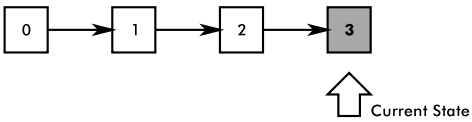
(c) The user edits the document, creating Version 4. While Versions 0 and 1 can be reached by invoking undo, Versions 2 and 3 are removed from the redo stack so they are unreachable.

Figure 3.9: How Stack-Based Undo loses information.

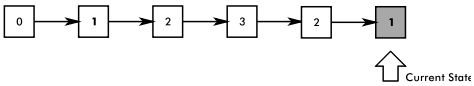
(Meyrowitz and van Dam, 1982; Archer et al., 1984), and currently it is most common form of undo: it is found in systems such as Microsoft Office, OpenOffice.org, Microsoft Internet Explorer, Apple Safari, Firefox, Adobe Photoshop, and the GIMP. A disadvantage of stack-based undo is that it uses more memory than bi-level undo. However, RAM is typically plentiful, so this should not be an issue.

Stack-based undo cannot be used to complete a branched error-recovery task (Section 2.4) as the redo stack is cleared after the user performs an action after doing an undo, as is illustrated in Figure 3.9. The inability to complete branched error-recovery using stack-based undo is similar to the problem that users experience when trying to return to pages using the back button when carrying out hub-and-spoke browsing (Figure 3.4). For optimal use, the user must also remember the sate of the document in each version, otherwise the user will have to assess whether the document is in the correct version after every undo action.

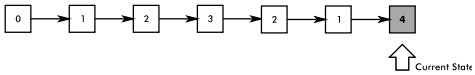
History undo originated in the INTERLISP environment in the mid-1970s (Kaisler, 1986) and is present in the Emacs family of text-editors (Harvey, 2003). Unlike stack-based undo, it allows the user to complete both branched and linear error-recovery tasks. Prior versions are appended to the list of



(a) The version of the document, 3, is erroneous, but the user can fix the error by reverting to Version 1.



(b) Undo is invoked twice and the document is reverted to Version 1.



(c) The user edits the document, creating Version 4. The system can still be reverted to all prior versions.

Figure 3.10: How history undo allows all past versions to be retrieved.

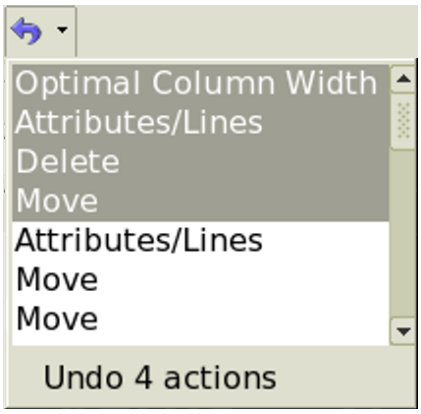
versions maintained by the system, without popping versions off a stack (Figure 3.10). Redo is not necessary in systems that employ history undo, as undo alone can be used to retrieve all prior document versions stored by the system. Despite the higher utility of history undo, Archer et al. (1984) claim that it is more confusing than stack-based undo.

3.3.1.2 Undo-Visualisation Systems

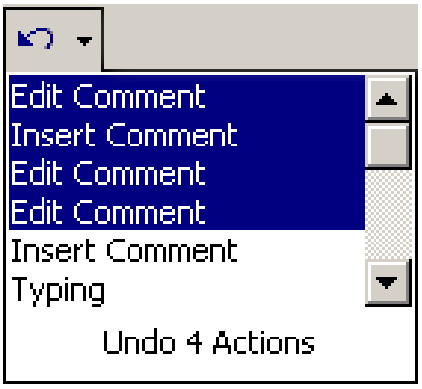
Standard stack-based undo and history undo systems generally provide little feedback when a new version of the document is created. Some systems provide the user with a visualisation of undo-data, which should make error-recovery using undo easier by providing cues to aid retrieval and recall, and making it clear when a new version has been created.

Applications in the Microsoft Office and OpenOffice.org suites present the user with an undo menu (Figure 3.11), which provide a visualisation of the recorded undo information. In both systems, when the user performs an action a new version is created, and the name of the action is appended to the menu. For example, Figure 3.11(a) shows that the last action to be performed in OpenOffice.org Calc was "Optimal Column Width". Selecting the item from the menu retrieves the version of the document that was current *just before* the selected action was carried out.

The raster-image editors Adobe Photoshop (Adobe, 2002) and the GIMP



(a) The undo menu in the OpenOffice.org Calc spreadsheet application.

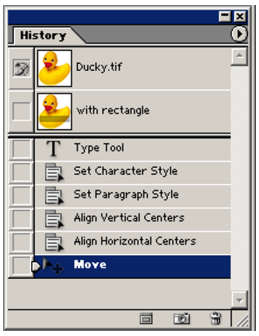


(b) The undo menu in the Microsoft PowerPoint presentation application.

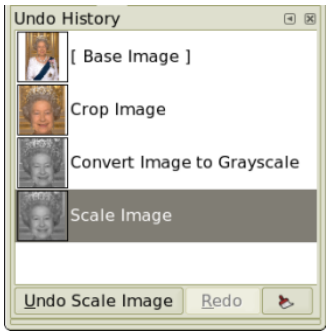
Figure 3.11: The undo menus in OpenOffice and Microsoft Office, showing how multiple actions, four in each case, can be undone. In both menus the most recent action is at the top.

(Gimp, 2005) also provide feedback so the user knows when a new version has been created. As the user carries out actions in Adobe Photoshop the names and icons of the completed actions are shown in the History Palette (Figure 3.12(a)). As with Microsoft Office and OpenOffice.org, the names of actions are associated with the different versions that are accessible by the system. While the keyboard can be used to invoke undo in Adobe Photoshop, as is standard, the user can also select an item in the History Palette. This causes the version of the document that was created by the selected action to become the current version. The GIMP provides a similar visualisation: its Undo History Dialog (Figure 3.12(b)). However, the GIMP differs from Photoshop in that a thumbnail of each version of the image is also placed in the list. The thumbnail acts as a description cue (Section 3.1).

The Flatland electronic whiteboard (Edwards et al., 2000; Mynatt et al., 2000) provides a different undo visualisation. Versions are placed on a simple timeline, rather than representing versions by actions and placing them in a list. Moving a slider left along the timeline will retrieve past versions, while moving the slider to the right will perform the equivalent of redo. The Flatland system seems to trade accuracy for speed: the position of the desired



(a) History Palette in Adobe Photoshop.



(b) The Undo History Dialog in the GIMP.

Figure 3.12: The visualisation of stack-based undo in the raster-image manipulation programs Adobe Photoshop and the GIMP. Both systems highlight the most recently completed action, located at the bottom of both images: `Move` and `Scale Image` respectively. Photoshop also lists two snapshots (`ducky.ttf` and `with rectangle`), while the GIMP displays a thumbnail of the document at each version in addition to the name of the actions.

version may not be exactly known, but the user should be able to change version more quickly than with standard undo systems.

3.3.1.3 Undo-Like Systems

In this section I will discuss a number of systems that are similar to the undo systems presented in Sections 3.3.1.1 and 3.3.1.2, as they also implicitly create versions when a user performs an action. I begin by looking at US&R, before examining multi-user undo systems. Finally, I discuss the Visage data visualisation system.

The user can carry out linear error-recovery and branched error-recovery using the command-line in the text-editing system US&R (undo, skip and redo: Vitter (1984a,b)). A user is also able to *manipulate* the sequence of actions that create the current document version, including altering the order of actions, removing actions and duplicating actions. This should increase the utility of the system. However, a mutable history is confusing, and runs contrary to the accepted physical and philosophical models of time (Priestley, 1964; Warren, 1988; Nahin, 1998; Grey, 1999). To my knowledge, there has been no study that has shown that the US&R system is usable.

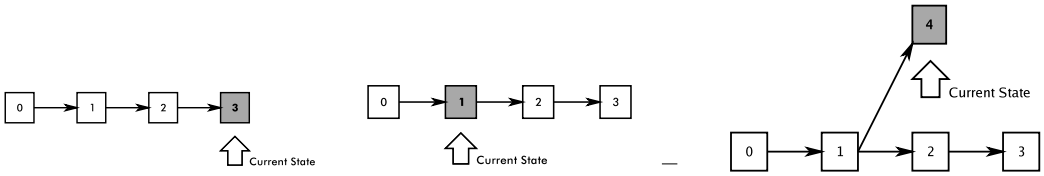
Research has examined how actions can be undone in multi-user (CSCW) environments where a user has the ability to edit the same document at the same time as others on the system (Prakash and Knister, 1992; Abowd and Dix, 1995; Wright, 1999; Sun, 2002; Carroll et al., 2003). In addition to the issues and limitations of single-user undo systems, undo in concurrent multi-user systems must address issues such as network-delays and whether a user can undo a change made by another user. Research into CSCW systems is beyond the scope of my thesis.

Unlike the systems discussed previously, which allowed the user to edit documents, the Visage system (Derthick and Roth, 2000) is used to visualise complex multi-dimensional datasets. Visage also comes equipped with an undo facility, which allows the user to return to a prior visualisation of the data. However, unlike the other undo systems discussed earlier, the undo data in Visage is kept in a tree, which makes it similar to the explicit version-creation systems that also tend to version store data in a tree (Section 3.3.2).

3.3.2 Explicit Version-Creation

Implicit version-creation systems tend to create many versions, creating a usability problem when searching for a specific version. Explicit version-creation systems overcome this problem by requiring the user to explicitly create a version. However, the user must remember to create a version at the appropriate point in the history of the document, or too much or too little of the document would be modified when linear or branched error-recovery is carried out. Adobe Photoshop provides a compromise: versions are automatically created when the user performs any action that alters the state of the document, but the user can also name important versions, creating 'snapshots' (Figure 3.12(a)).

In this section I will discuss two common types of explicit version-creation systems: version-control systems that store a tree of document versions (Section 3.3.2.1) and history-based version retrieval systems that maintain the document versions in a temporal list (Section 3.3.2.2).



(a) The current version of the document (3) is erroneous, but the user can fix the error by reverting to Version 1.

(b) Undo is invoked and the document is reverted to Version 1.

(c) The user edits the document, creating Version 4. All prior versions can be reached by selecting the desired version from the tree.

Figure 3.13: Creating a branch using Tree-Based Error Recovery (Based on a diagram by Mynatt et al. (2000)).

3.3.2.1 Version-Control Systems

There are many different different version-control systems, such as the Revision Control System (RCS: Tichy, 1985), Concurrent Versioning System (CVS: Fogel, 1999), Bitkeeper (Kroah-Hartman, 2002), Microsoft SourceSafe (Microsoft Corporation, 2002), and Subversion (Collins-Sussman et al., 2004). There is "considerable industrial experience" with such systems for managing different versions of source-code for software (Krühaut and Zeller, 1999), and the aforementioned systems are generally used by software developers to control access to source code (rather than direct interaction with the file-system). However, it is possible to use version control systems to store and retrieve documents other than source code (Hess, 2002, 2005).

In this section I will examine how version-control systems organise versions in a tree (rather than a linear data structure like most undo systems), the cues that can be used to retrieve versions, and how version-control systems require the user to explicitly store versions of a document.

Version control systems store versions in a tree (Figure 3.13), unlike bi-level undo, stack-based undo and history undo, which store versions in a linear data structure. The use of a tree is important for two reasons. First, branched error-recovery can be completed in version-control systems: the user selects the version from an alternate branch, if required. Second, a tree allows versions of a document to be modified independently of each other.

yelp-index-model.h	**1.5**	2 years	hallski	2003-06-13 Mikael Hallendal <micke@imendio.com> * *: Updated my conta...
yelp-info-pager.c	**1.11**	4 weeks	chpe	2005-05-16 Christian Persch <chpe@cvs.gnome.org> * src/yelp-base.c: (yelp_ba...
yelp-info-pager.h	**1.1**	5 months	shaunm	* src/Makefile.am: * src/yelp-info-pager.c: * src/yelp-info-pager.h: * src/yelp-...

Figure 3.14: A hypertext view for the CVS repository for the GNOME help viewer: Yelp. It shows the head-version of the source-code files used to create the program. The cues available in this view are (from left to right) the title of the file (which also incorporates the ID and format), version number of the file (relation), age of the version, the creator of the version, and a description of the last change.

This allows an author to work on a version of a document for one audience while also working on a different version of the same document for a different audience. This is important for software developers, who often have to add features to one (unreleased) version of the software while also being able to fix bugs in an older version of the software. However, the use of a tree may be more confusing than a linear system such as undo.

While graphical views of version-retrieval systems are possible, Figure 3.14 shows a hypertext view of a CVS store, with the typical cues that are available from a version-control systems: title, version number, age, creator and description. When CVS is used for document retrieval, the name is used to form the retrieval cue. For version-retrieval, the version number specifies the relationship between the current version (often called the 'head') and the other versions. Increasing numbers indicate increasing versions, as they do in Figure 3.13; successive decimal points are added to the version number for each branch — so, in Figure 3.13, Version 4 would be called Version 1.1. Like the age and creator of the file, the version number can be assigned automatically by the system. However, the description of the version must be added manually when the version is stored.

Many of the cues used by version-control systems, such as the file name and description, have to be supplied by the user when the version is explicitly added to the system. However, the user of the Historian system (Abu-Shakra and Fisher, 1998) does not have to supply any cues, because the system only supplys one cue to document retrieval: relation. Historian is an extension of

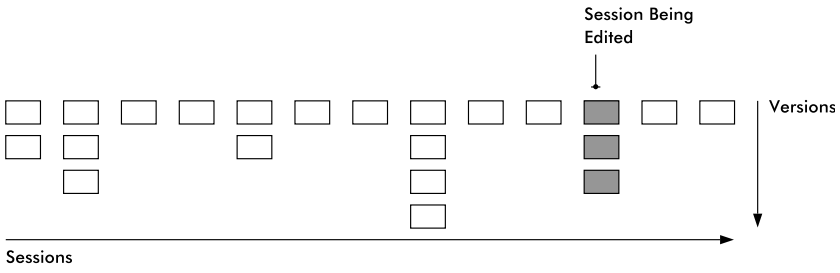


Figure 3.15: The visualisation of the implicit versions in the Historian system (Abu-Shakra and Fisher, 1998). Sessions are created when the user starts the editor, and versions are created whenever the user saves.

the Emacs text-editor that creates versions whenever the user saves. Versions are grouped by the editing 'session', which is defined as a period during which the editor was running (Figure 3.15). Each time the user saves, a version (represented by a box) is added to the bottom of the session-stack. Like other version-control systems, Historian presents a tree of versions and the user can manipulate each branch independently. To my knowledge, no other version-control system that has implemented branching in the same way as Historian.

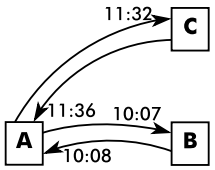
3.3.2.2 History-Based Version Retrieval

History-based version retrieval systems organise document versions in the order that they are created, rather than organising the versions in a tree as is done with version-control systems. This linear organisation of document versions is similar to how actions are organised with history undo (Section 3.3.1.1). There are also similarities with the temporal back button (Section 3.2.5.1) and history list systems (Section 3.2.5.2) that also organise documents in a linear temporal order (albeit different documents, rather than versions of the same document).

The earliest history-based version retrieval system that I am aware of is the Programmers Utility Filing System (Wilkes, 1964), which was an experimental file system that had version-control but no ability to name documents. More widely known, the VMS operating system (Compaq, 1999) implements a history-based versioning system by appending a version-number to the end

(cur) (last) 11:36, 18 Jun 2005 David Newton **m** *(Reverted edits by 67.15.54.56 to last version by Znode)*
(cur) (last) 11:32, 18 Jun 2005 67.15.54.56
(cur) (last) 10:08, 18 Jun 2005 Znode *(rv vandal)*
(cur) (last) 10:07, 18 Jun 2005 69.203.114.231 *(→History)*
(cur) (last) 09:57, 18 Jun 2005 Ahoerstemeier **m** *(Reverted edits by 69.203.114.231 to last version by 1pezguy)*

(a) The cues for version retrieval are similar to those presented by CVS (Figure 3.14): date, user and description. The 'curr' and 'last' links are for comparing versions of a document.



(b) A tree representation of the versions shown in Figure 3.16(a).

Figure 3.16: The history of changes to the 'Warren County Canal' page on the Wikipedia system (WikiMedia Foundation, 2005). Anonymous users carried out two editing actions on the page at 10:07 and 11:32, to create Versions B and C shown in Figure 3.16(b). Both of these editing actions were reverted, by the users Znode and David Newton at 10:08 and 11:36 respectively, to return to the original version of the document (Version A).

of each file name when the user saves the document. Some wiki systems also store each version of a document, and present this information to the user as a history list. Figure 3.16 shows a history for a page in Wikipedia, which uses the MediaWiki system (WikiMedia, 2005) to organise documents. Unlike VMS, dates are used to distinguish between versions, rather than version numbers.

To reduce the number of versions that have to be managed, the Elephant File System (Santry et al., 1999) deletes some old versions after a period: the system 'forgets' versions. The Lifestreams system (Freeman, 1997) reduces the number of versions in a different way from the Elephant File System: it overwrites the prior version if it is less than a day old, thereby relying on the temporal behaviour of the user to indicate when significant versions are created.

3.3.3 Version Retrieval: Conclusion

The following are guidelines for the implementation of version retrieval systems.

- Bi-level undo, as found in systems such as vi, should not be implemented: stack-based undo, history undo, or a tree-based undo system

should be implemented in preference.

- If versions will be edited independently then a tree-based error-recovery system (such as version-control) is needed, rather than a version history or stack-based undo.

- An undo visualisation could be provided, such as that found in Microsoft Office or Adobe Photoshop, as it may assist users in carrying out error recovery.

- If a large number of simple actions are stored by the system, then a graphical slider may be an appropriate alternative to the using the keyboard, menu or toolbar button, as demonstrated by the Flatland system.

- Requiring the user to explicitly create a version may reduce the utility of the system, but fewer versions should make retrieval easier if the user created versions at appropriate points in the past. Designers will need to decide which of these conflicting requirements is more important for a system, or attempt to combine both as is done in Adobe Photoshop.

A number of questions are raised from the prior work into version retrieval systems. First, why is history undo — which appears to have higher utility than stack-based undo — not more widely implemented? Archer et al. (1984) claim that is because it is more confusing than stack-based undo, but do not know of any empirical evidence to support the claim. Second, why are tree-based systems not more widely implemented outside version-control systems? In Chapter 5 I present the results of an empirical evaluation of different types of undo, including stack-based undo and tree-based undo, to determine if a tree-based system has any usability issues to prevent it being used more. Then, in Chapter 6, I evaluate a system that integrates explicit and implicit version creation. In these studies I show that there are no usability reasons for tree-based error-recovery systems to be rare.

3.4 Conclusion

I began this chapter by introducing some cues that are used by document retrieval systems, with particular attention given to the cues defined by the Dublin Core Metadata Initiative and how they are supported by various systems. Next, I looked at several temporal document-organisation systems, and how they supported the finding, reminding and storing tasks. Unlike many systems, temporal document-organisation automatically creates a relationship between documents, allows the documents to act as reminders of the user's current tasks, and allows the recently edited documents to be found quickly and easily. Using version retrieval to correct errors or explore system state was then discussed. Various version-retrieval systems were examined as part of the discussion, including various types of undo, undo visualisations and different types of version-control system.

The findings presented in this chapter will be used to inform the design of the empirical evaluations of history-lists (Chapter 4) and error-recovery methods (Chapter 5). The findings from this chapter, and the following two chapters, will then be used to inform the design of a system that combines temporal document-organisation and version organisation in the same interface (Chapter 6).

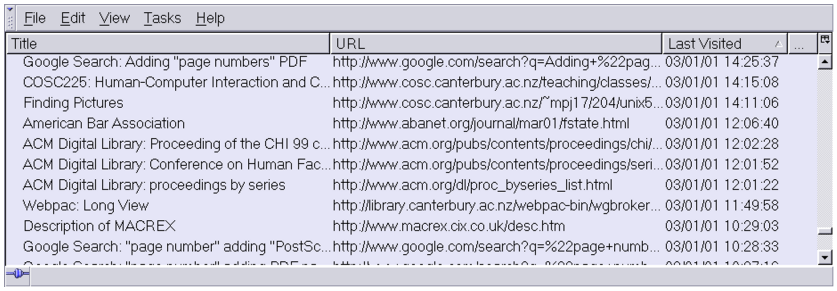
Chapter IV

Evaluation of History Lists

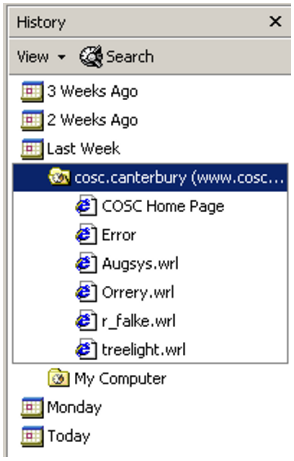
In this chapter, I present the results of a controlled empirical evaluation of document retrieval from history lists. The experiment is based around the retrieval of Web pages from four lists: the first simply presents documents in a temporal order, the second uses colours to highlight different 'temporal chunks' in the list, the third interface shows only the pages that were viewed during the selected chunk, while the final interface introduces categories that further break up the temporal chunks. I found that retrieval is fastest from the interface that shows only the pages that were viewed on the selected chunk. I also found that retrieval is generally slower when the documents are categorised.

The results of the experiment presented in this chapter can be used to inform the design of simple document histories — so the study belongs to the first of my research themes: the retrieval of documents that are organised by time. The design of the interfaces in this experiment was informed by the guidelines presented in Chapters 2 and 3. In addition, the results are used to inform a system that combines document and version retrieval, which is presented in Chapter 6.

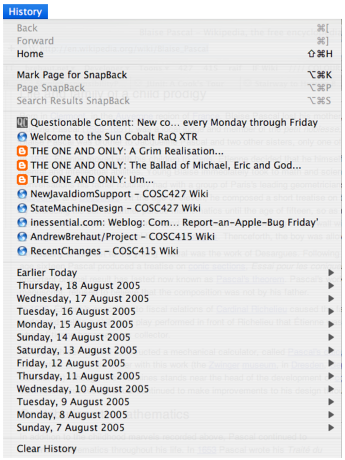
History lists are a temporal document-organisation interface that is commonly implemented (Section 3.2.5.2). To my knowledge there has been no other empirical study that has sought to determine which temporal document organisation scheme is better: a simple list (such as that found in Navigator, Figure 4.1(a)), a categorisation (like Internet Explorer, Figure 4.1(b)), or chunking (like Safari, Figure 4.1(c)). The aim of this experiment is to assess the benefits of adding features to history lists, such as categorisations and temporal chunks, and to create guidelines for their implementation. While Web-centric, the results from this experiment should be applicable to interfaces outside the information foraging domain.



(a) The History Window of Netscape Navigator 4, with the user's browsing history shown as linear list.



(b) The History Pane of Microsoft Internet Explorer 5, with the history broken down by temporal chunk and site.



(c) The History Menu of Apple Safari 2 with the history broken into chunks.

Figure 4.1: The history lists of Netscape Navigator 4, Apple Safari 2 and Microsoft Internet Explorer 5.

I started this evaluation with the belief that categorising documents in a temporal interface would not allow for faster retrieval, as searching should be more difficult if the user does not know the category that the document is stored under. However, I believed that breaking the history into smaller parts, such as using colours to delineate the days, would lead to faster retrieval times, as temporal searching would be easier.

The method of evaluation is presented first (Section 4.1), with the results, the experimental concerns and discussion following (Sections 4.2–4.4). This chapter is concluded in Section 4.5.

4.1 Method

The experiment compared how quickly participants retrieve Web pages from four history list interfaces, in order to gain an insight into the relative performance of the different interfaces as the retrieval cue is varied.

The design of the experiment is discussed in Section 4.1.1. This is followed by a presentation of the four interfaces that were tested (Section 4.1.2) and the cuing conditions that were used to prompt the participants' retrievals (Section 4.1.3). The participant details and treatment is detailed in Section 4.1.4.

4.1.1 Design

The experiment was designed as a three-factor repeated-measures analysis of variance (ANOVA) for factors interface-type, date-cue and title-cue; task-completion time was the dependant measure. The interface-type factor had four levels: Linear, Chunking, Two Pane and Tree (Section 4.1.2). The two cuing factors had two levels each: precise and vague (Section 4.1.3).

4.1.2 Interfaces

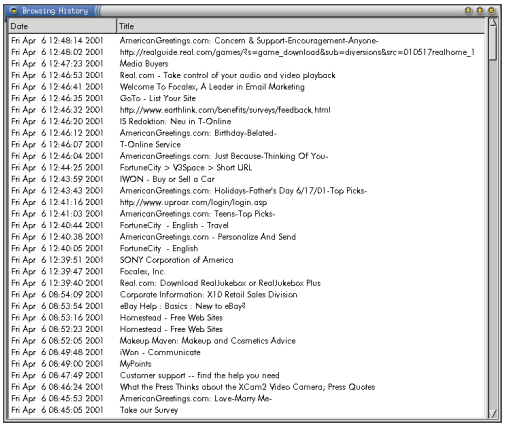
The experiment tested four history list interfaces (Figure 4.2). The simplest is a linear list, based on the history list in Navigator, which has the smallest number of widgets, and does not break up the temporal list (Figure 4.2(a)). The next most complex is the chunking interface that uses colours to group

	Navigator	Safari	Internet Explorer
Title	✔	✔	✔
Creator			
Subject			
Description			
Date	✔	✔	✔
Format			✔
ID	✔		

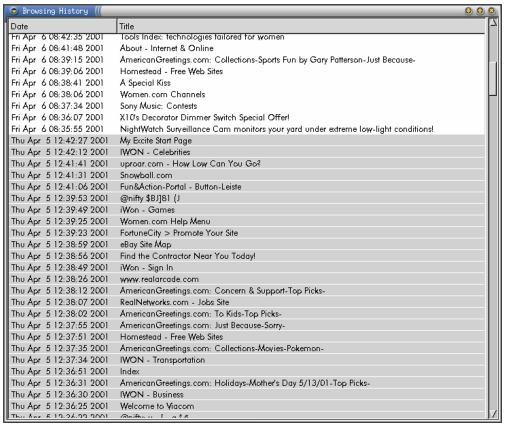
Table 4.1: The cues, introduced in Section 3.1.2, that the history list interfaces in Netscape Navigator 4, Apple Safari 2 and Microsoft Internet Explorer 5 support.

pages into days, but is otherwise the same as the linear list (Figure 4.2(b)). The two-pane interface (Figure 4.2(c)) also splits the temporal list of documents into days, but the user must select the day that is shown. Finally, the most complex interface is the categorised tree-based history, based on Internet Explorer, that broke the list of documents into days and then categorised the pages that were viewed on the selected day (Figure 4.2(d)). To control potential confounding factors, the cues provided by the interfaces were limited to the document title, the date that the document was last visited, and the relationship between documents. These cues are common to the history lists found in the most widely used Web browsers (Tables 3.3 and 4.1).

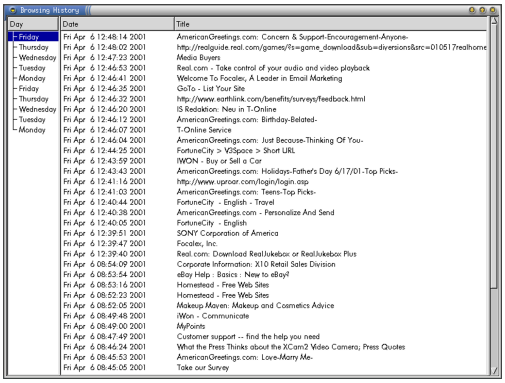
The interface with the fewest controls is the Linear List (Figure 4.2(a)), which was based on the history window of Netscape Navigator 4. It consists of a simple list and a scrollbar, with the most recently viewed page at the top. As with all interfaces, three cues were provided: the date the page was last viewed is displayed on the left-hand side of the list, the title is shown in the second column and the order of the documents formed the relation cue. Retrieval from the Linear interface should be fast when the date the page was viewed is known, as the user can scroll to quickly locate the correct date.



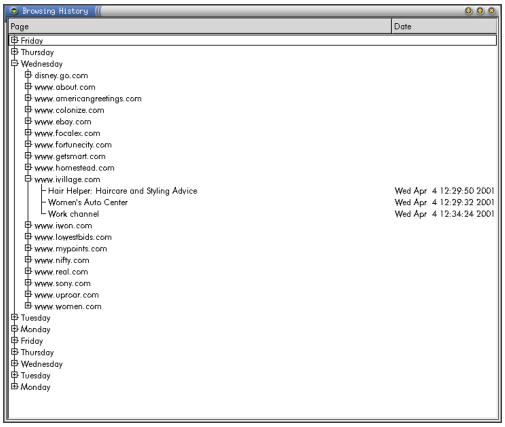
(a) The Linear-List interface sorted the pages with the most recently viewed at the top.



(b) The Chunking interface was the same as the Linear List except the background colour was changed when the day changed.



(c) The Two Pane interface divides the list into days, shown in the left-hand pane, with all the pages viewed on that day shown on the right.



(d) The Tree interface divides the list into days and then each day is divided further into sites. Within each site the pages are sorted alphabetically by title

Figure 4.2: The four history list interfaces used in the experiment.

However, trying to find one page out of the many displayed may prove to be difficult and slow.

The most complex interface is the Tree (Figure 4.2(d)), which was based on the history pane of Microsoft Internet Explorer 5. The Tree has three levels:

1. The day the page was viewed,

2. The site the page was from (with the pages alphabetically listed under each site) and

3. The leaf nodes of the tree represent the actual pages, and were sorted alphabetically.

Each page entry consists of the title on the left, and the date the page was viewed on the right. The date cue was one of the two substantive differences between the Tree and the history list in Internet Explorer, as the latter does not provide the user with a detailed date. The other major difference is that all temporal chunks are days in the experimental interface, rather than days and weeks. To retrieve a page the user has to click on each level in the tree to view deeper in the tree, compared to simply scrolling in the Linear interface. Retrieval from the Tree should be slower than retrieval from the Linear interface, as the site-classification breaks the temporal ordering, making it harder to find a page using date as a cue. However, retrievals from the Tree should be fast if the user knows the site from which page originated, as the correct site is easier to find using the Tree than the Linear interface. In addition, there are fewer items visible in the Tree, as only one branch can be open any one time, so participants should find it easier to search the Tree interface.

Two other interfaces were developed so features in the spectrum between Linear and Tree could be tested. To see if splitting the interface into days effected performance a Chunking interface was created (Figure 4.2(b)). To increase the *perception* of days, pages that were viewed on the same day are highlighted, by changing the background colour of the list between white and tan. This is similar way to the Netscan newsgroup visualisation system highlights chunks (Smith and Fiore, 2001). Otherwise, the Chunking interface is

the same as the Linear interface. The Chunking interface was expected to be faster than the Linear List because days can be seen more clearly.

The final interface is the Two Pane list (Figure 4.2(c)). In this interface, the browsing history is split into days, with the days listed on the left-hand side of the window. Selecting a day shows all pages that were viewed on that day in the right-hand list, with the most recently viewed page at the top. The Two Pane list is similar to the History menu in Apple Safari 2 (Figure 4.1(c)) but a window was used, rather than cascading menus. The only major difference between the Two Pane and Tree interfaces is that the Tree further classifies pages by site, rather than temporal order. Retrievals from the Two Pane interface were expected to be slower than the Linear and Chunking interfaces when the user knows the date of a page view, as the user has to click on a day and scroll to find a page rather than simply scrolling.

The size of all the interfaces was 722×529 pixels, and all were displayed on a full-colour monitor with the resolution of 1600×1200 pixels. Each interface held an artificial browsing history consisting of 373 pages split over ten days (two working weeks), which is slightly lower than the 41.9 page-views per day that was reported as the mean by Cockburn and McKenzie (2001). Over a thousand pages were randomly selected from the 'Global Top 50' list of Websites by Jupiter Media Metrix (2001), and four randomly chosen subsets of these pages were used to populate the interfaces. The artificial browsing history is the source of an experimental concern, which is discussed in Section 4.3.

4.1.3 Cuing

Retrievals were cued with the date that the page was viewed and the title of the page. The date cue had two levels: precise, with the exact date given to the participant, or vague, with a range of three to four days provided. The title cue could also be precise or vague. A precise title was the same as the title in the history list, and included the site name; a vague title consisted of a keyword that was taken from the full title, and did not include the site name. Table 4.2 lists the four possible cuing conditions, with examples.

Cuing Condition	Example	
	Title	**Date**
Precise Title, Precise date	Lots of Lemons at Fruit.com	Wed Apr 4 12:34:24 2001
Vague Title, Precise date	Lemons	Wed Apr 4 12:34:24 2001
Precise Title, Vague date	Lots of Lemons at Fruit.com	One to three days ago
Vague Title, Vague date	Lemons	One to three days ago

Table 4.2: Example cuing conditions.

4.1.4 Participant Details and Treatment

Eighteen third-year Computer Science students participated in the experiment, and were rewarded with a lottery ticket. Microsoft Windows was the 'normal' operating system for five participants, Unix was normal for four, while nine said that both Windows and Unix were their normal operating systems. The participants' normal browsers followed a similar split: five participants normally used Microsoft Internet Explorer, another five used Netscape Navigator, while the remaining eight normally used both Internet Explorer and Navigator. (Unix, with Netscape Navigator, was the standard computing environment used by third-year Computer Science students at the University of Canterbury at the time of the experiment.)

The experiment started with the purpose of the evaluation being explained to the participant. Each participant used all the interfaces, with the presentation order balanced, to mitigate any learning effects. In addition, the pages listed in each interface were changed to further avoid learning effects. The page view-times were not changed between interfaces, so, for example, all interfaces had a page viewed at 'Fri Apr 6 08:49:00 2001'. (No participants realised that the page view-times were the same between interfaces.) When an interface was introduced, the participant was shown how to retrieve pages using the interface. A single training task was given: to retrieve the most recently viewed page in the history, which was the top-most item in all the interfaces except the Tree. Each participant was then asked to retrieve eight different pages, two for each of the cuing conditions. Timing of the retrieval task, which was automatically logged, was started when the cue was given and stopped when the user clicked on the correct page.

After retrieving the page the participants were asked if they agreed with

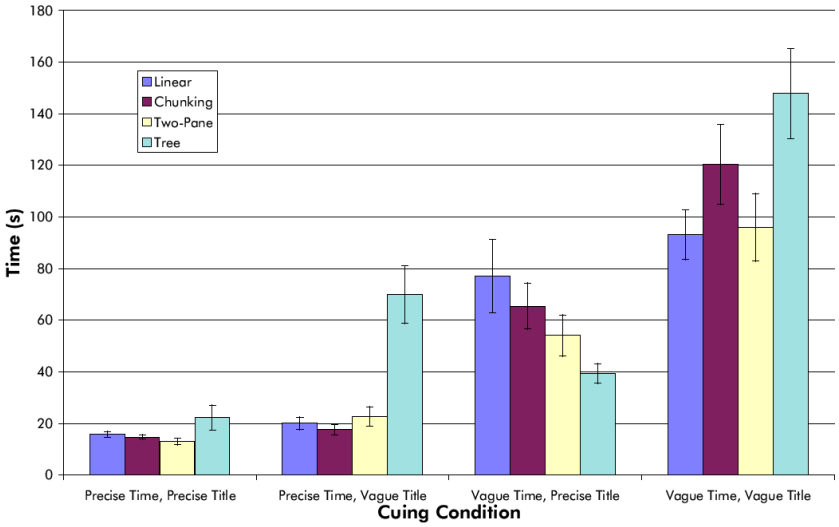


Figure 4.3: Mean page retrieval times for each interface under each cuing condition. Error bars indicate ± one standard error around the mean.

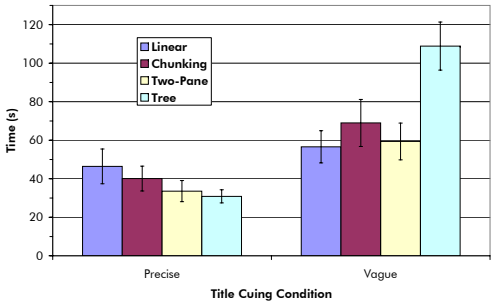
the statement "It was easy to find this page", using a five-point Likert scale, with five as 'agree'.[1] At the end of the experiment the participant was shown a picture of each of the interfaces in turn and asked if he or she agreed to the two statements "This interface was easy to use" and "I could find pages quickly using this interface", as rated on the same five-point Likert scale as before. For the final questions the interfaces were shown from simplest to most complex: Linear List, Chunking, Two Pane and Tree.

4.2 Results

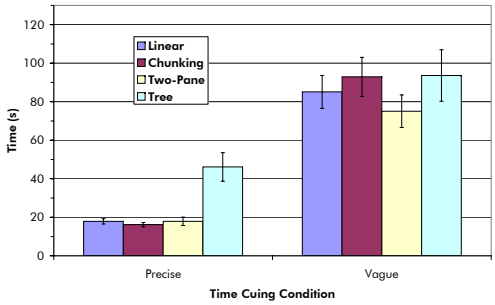
All the participants understood and successfully completed the experiment. The mean time taken to retrieve each document was 55.6s ($\sigma = 53.4$). The high variation in times can be explained by the substantial difference between precise and vague cuing times, as shown in Figure 4.3.

There was as significant difference between the mean retrieval times for the four interfaces ($F(3, 42) = 4.8$, $p < 0.01$). Users took 69.8s ($\sigma = 63.4$) to retrieve pages using the Tree, which was the slowest interface overall. The

[1] Course surveys at the University of Canterbury, which most students fill out regularly, use a five-point Likert scale.



(a) Mean page retrieval times for each interface under the two title cuing-conditions.



(b) Mean page retrieval times for each interface under the two date cuing-conditions.

Figure 4.4: Mean page retrieval times across interface-type and cuing condition.

fastest retrievals were with the Two Pane interface, which took a mean time of 46.5s ($\sigma = 44.2$). The mean retrieval times using the Linear and Chunking interfaces were 51.5s ($\sigma = 47.5$) and 54.5s ($\sigma = 55.0$) respectively. Post-hoc analysis shows there was a significant-difference between the retrieval times from the Tree and Two Pane interfaces, as well as the Tree and Linear-list (Tukey Test, HSD = 17.5). However, there was no honest significant-difference between any other pair of means.

Retrievals were significantly faster in the precise-title cuing condition, compared to the vague-title condition ($F(1, 14) = 63.5$, $p < 0.01$). The mean time taken by participants to retrieve a page, when the exact name of the page was given, was 37.7s ($\sigma = 62.1$); retrievals in the vague-title cuing condition took a mean time of 73.5s ($\sigma = 62.1$).

There was also a significant difference between document retrieval times under the precise and vague-date cuing conditions ($F(1, 14) = 232.4$, $p < 0.01$). When the precise-date was given as a cue, participants took a mean time of 24.5s ($\sigma = 24.9$) to retrieve Web pages, compared to 86.7s ($\sigma = 56.3$) for the vague-date cuing condition.

There was a significant interaction between the independent variables interface and title cuing-condition ($F(3, 42) = 11.7$, $p < 0.01$), as can be seen in Figure 4.4(a). The slowest retrievals were when the Tree interface was used with the vague-title cue: a mean time of 108.9s ($\sigma = 68.4$). The

next slowest interface under the same cue was Chunking, which took 69.0s ($\sigma = 67.0$). The fastest mean retrieval-time, of 30.9s ($\sigma = 18.6$), was with the Tree interface under the precise-title cuing condition, 2.7s faster than the next-fastest interface: Two Pane. The slowest retrievals with the same cue were with the Linear interface (46.4s, $\sigma = 49.4$).

There was no significant interaction between the date cuing-condition and the interface independent variables ($F(3, 42) = 2.4$, $p = 0.08$). The fastest retrievals under the precise-date cuing condition were with the Chunking interface: 16.2s ($\sigma = 6.2$) as can be seen in Figure 4.4(b). The slowest mean-retrieval time under the same cuing condition was 46.1s ($\sigma = 40.7$), with the Tree. Under the vague-date cuing condition, the slowest mean-retrieval time was with the Tree interface, at 93.6s ($\sigma = 73.1$). While the mean retrieval times with the Two-Pane interface, which was the fastest interface under the same cuing condition, took 75.0s ($\sigma = 46.4$).

There was a significant interaction between the date and title cuing conditions ($F(1, 14) = 28.1$, $p < 0.01$). The fastest retrievals were when both cues were precise, with a mean time of 16.5s ($\sigma = 10.4$); the slowest retrievals were when both cues were vague (114.3s, $\sigma = 58.0$). Retrievals when precise-date and vague-title cues were given took a mean of 32.6s ($\sigma = 31.7$), while retrievals cued with a vague-date and a precise-title took a mean of 59.0s ($\sigma = 38.4$).

There was no significant interaction between the interface, title-cue and date-cue independent variables ($F(3, 42) = 2.4$, $p = 0.8$). As can be seen in Figure 4.3, the fastest retrievals were with the Two-Pane interface when both cues were precise (13.0s, $\sigma = 5.0$), while the slowest retrievals were with the Tree interface when both cues were vague (147.8s, $\sigma = 67.6$). The Linear, Chunking and Two-Pane interfaces had similar retrieval times under the two precise-date cuing conditions, while retrievals with the Tree interface were slower. Under the precise-date precise-title cuing conditions, retrievals from the Tree interface took 22.3s ($\sigma = 18.5$) compared to 13.0–15.7s for the other three interfaces. The difference between the Tree and the other three interfaces was greatest under the precise-date vague-title cuing conditions: retrievals from the Tree took a mean time of 69.9s ($\sigma = 43.2$), while retrievals from the other three interfaces took 17.6–22.7s. However, retrievals from the

Linear	Chunking	Two Pane	Tree	Friedman Test	$p < 0.05$
This interface was easy to use.					
3.1 (0.8)	3.4 (0.8)	3.8 (1.0)	2.2 (0.9)	$X_r^2 = 18.2$, DF $= 3$	✔
I could find pages quickly using this interface.					
3.8 (1.1)	4.0 (0.8)	4.1 (0.8)	2.6 (1.1)	$X_r^2 = 23.2$, DF $= 3$	✔
It was easy to find this page.					
3.4 (1.4)	3.4 (1.3)	3.6 (1.2)	2.9 (1.4)	$X_r^2 = 22.6$, DF $= 3$	✔

Table 4.3: The mean responses to the three Likert-scale questions (five is agree) with the standard deviation given in parenthesis.

Tree interface were faster than the other interfaces under the vague-date precise-title cuing conditions: participants took a mean time of 39.4s (σ = 14.7) to retrieve pages, compared to 77.1s ($\sigma = 54.9$) for the Linear interface, which had the slowest retrievals. Participants were slower at retrieving pages from the Tree interface than the other three interfaces under the vague-date vague-title cuing condition, taking a mean time of 147.8s ($\sigma = 67.6$), compared to 93.1s ($\sigma = 37.3$) for the Linear interface, 95.6s ($\sigma = 50.7$) for the Two-Pane, and 102.3s ($\sigma = 59.8$) for the Chunking interface.

4.2.1 Subjective Measures

Users thought retrievals from the Two Pane interface were easier and quicker than those from the other three interfaces — as can be seen in Table 4.3 — and mean rating for the Tree interface was lower than the other interfaces for all three questions. While the Chunking interface rated at or above the ratings for the Linear interface, the ratings for the two interfaces were within a standard deviation.

At the end of the experiment, the participants were asked to rank the interfaces. The Tree had the lowest mean ranking at 3.5 ($\sigma = 0.9$), while the Two Pane interface had the highest ranking at 1.2 ($\sigma = 0.6$). The Linear List and Chunking interfaces ranked 3.1 ($\sigma = 0.6$) and 2.1 ($\sigma = 0.6$) respectively.

4.2.2 Participant Comments

The time to complete the retrieval tasks when both cues were vague caused participants to become frustrated, with the comment "That was horrible"

being representative. Participants liked the precise-date cuing condition: typical comments were "Specific times helped", "Time was a handy tool", and "Anything with a date gets a five".

The participants often said that the Linear List was a simple interface that did not confuse. Typical comments were "Easy, but it took a while", "It was just a scrollbar", and "There is not much you can do other than scroll with this one." Some participants commented that it was possible to scroll too far when performing a vague-date search, with comments such as "I've gone way too far" and "Oh, I went too far." The amount of scrolling needed to retrieve a page also caused some to complain, with one participant exclaiming "I hate scrolling."

Many participants did appreciate the use of colours in the Chunking interface, with many comments such as "I liked the colour things", "Colours did help" and "I think the colours help." However one participant did state that "Colours did not make a difference to how I was searching", and others stated that "Colours [were] not so helpful" and "Colour distracted me slightly." As with the Linear interface, the need for scrolling prompted some negative comments, with one participant stating that scrolling was a "pain."

Participants tended to like how the Two Pane interface split the list into days, prompting comments such as "Probably the fastest", "Easy to navigate", and "The days helped heaps." However, the use of just the day names did cause some problems, with one participant mentioning how matching the day to a date was difficult and another commented that "I clicked on the wrong Monday to start with."

The comments made by the participants about the Tree interface reflected the low scores in the subjective measures (Section 4.2.1). Some typical comments were "Ah horrible!", "Still don't like it" and "Well, that was confusing." Participants often stated that if the URI could be determined from the title-cue then searching could be fast, otherwise the "Title was useless."

4.3 Experimental Concerns

There are a number of concerns with the experiment. First, the user had no prior knowledge of when a page was viewed — so the cue had to be relied on

to provide all the information. I suspect that this caused retrievals in this experiment to be slower than those from user's own history, but it did not create a bias for any particular interface.

Second, there was no context for page items, which reduced the effectiveness of the relationship cue. Histories typically have clusters of related documents, or 'episodes' (Section 3.2.5.2). Therefore, the user can look for pages that belong to an episode rather than a single page; this was not possible in the experiment. This should lead to slower retrieval times from all interfaces except the Tree, which does not display items in a strict temporal order.

Finally, the participants may not have been familiar with the sites in the history, so when a vague title cue, such as 'more music', was provided, the participant did not associate the correct site with the keywords. This may have lead to slower retrievals from the Tree interface, under the precise title cuing condition, as the user must know the site that is associated with a page for quick retrieval.

4.4 Discussion

The retrieval times in the experiment are notable for three features:

1. The similarity of retrieval times between the Linear, Chunking and Two-Pane interfaces,

2. The Tree interface is a frequent outlier, and

3. The amount of data provided to the participants has a large effect in retrieval times.

Retrieval times from the interfaces that kept the Web pages in a strict temporal order — the Linear List, Chunking and Two-Pane — are very similar, as can be seen in Figures 4.3 and 4.4. While retrievals using the Two-Pane interface are faster overall, there is no honest significant-difference between the Two-Pane and the other two strictly-temporal interfaces. There is a tendency for the Two-Pane to be faster in retrievals that are cued with precise titles, but overall the statistical error is too large to draw any conclusions

from this. The mean retrieval times for the Chunking interface are 24.3s slower than those for the Two-Pane interface in the vague-date vague-title cuing condition, but I conclude that this is due to statistical error. However, the participants' comments and subjective measures show that they prefer the Two-Pane interface to the others.

In contrast to the other three interfaces, the Tree is frequently an outlier in the results. It is 39.9s slower than the next-slowest interface when retrievals are cued with a vague title (Figure 4.4(a)), and 28.2s slower than the next-slowest interface when retrievals are cued with a precise date (Figure 4.4(b)). However, with precise titles, the Tree is faster than the other interfaces, but not significantly so. Looking at the interaction between interface, title cue and date cue (Figure 4.3) we can see that generally the Tree is the slowest interface, with up to a 47.2s difference between the mean retrieval times from the Tree compared to the next slowest interface (under the precise-date vague-title cuing condition). However, when retrieval is cued with a vague date *and* a precise title the Tree is the fastest interface, but not greatly so. Participants indicated that they found precise-title retrievals easy with the tree because they could use the title to determine the site from which the page was from. Comments also show that the opposite is also true: retrieval is slow when the site cannot be determined from the title.

As expected, the amount of information provided as a cue has a large affect on the speed of retrievals. The date cuing-condition has a bigger affect on retrieval speed than the title cue, with a 62.1s difference between vague and precise retrieval times, compared to a 35.7s difference between the title cues. I suspect the difference is due to the ease of searching for pages when given a precise date: the documents in the Linear, Chunking and Two-Pane interfaces are sorted by date, so they were easy to search when an exact date was given. Once again, the participants' comments back this hypothesis.

4.5 Conclusion

In this chapter I presented an evaluation of four history lists. I showed that retrieval from a history list that employed categories, such as that found in Microsoft Internet Explorer, is slightly faster than retrieval from other

interfaces if the user knows the category that the page belongs to, but the difference in retrieval times is not statically significant. Overall, participants preferred the interface that breaks the history into days — and did not use a second-level category — even though retrieval was not statically quicker than the other two strictly-temporal interfaces.

Some guidelines can be drawn from the results of the evaluation.

- A classification should be used if the user *does* know the category that the document is organised under, as retrieval will be fast.

- If the user *does not* know the classification, then the usability of the history list will be severely effected if a classification is used to organise the documents: retrieval times will be slow, and the user will become frustrated.

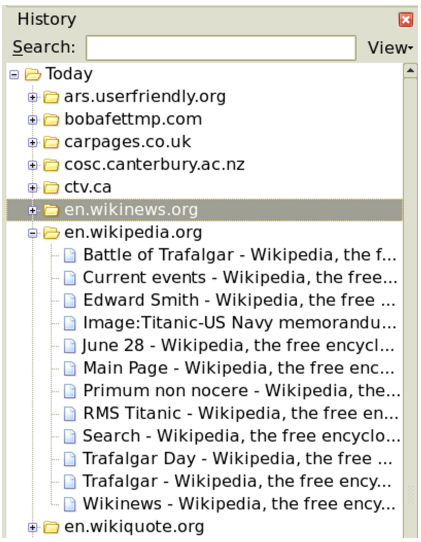
- The conservative option would be for a history list to be divided into temporal groups, as is done with the Chunking and Two Pane interfaces. Neither interface allows the user to retrieve the pages in the shortest possible time under all cues, but both satisfy the principle *primum non nocere* 'first do no harm', as neither is the slowest interface. In addition, participants preferred the Two Pane interface.

- Ideally, the user should be able to choose between classified and unclassified views of the history, as is done with the history list in Mozilla Firefox 1, the successor to Netscape Navigator 4 (Figure 4.5).

This study was undertaken to attain an initial answer to one of my primary research questions: what is the best way to use time to organise different documents. The evaluation was a success as the results did indicate that the Chunking and Two-Pane are good, while the Tree preformed poorly overall. However, the evaluation did not provide a categorical answer to the research question for four reasons.

- While a number of common interfaces were evaluated, an exhaustive test of all interfaces was not presented.

(a) The 'Last Visited' view presents the pages in the same order as those in Netscape Navigator, but only the title and relation cues are provided.



(b) The 'Date and Site' view presents the pages in the same order as those in Microsoft Internet Explorer.

Figure 4.5: The history panel in Mozilla Firefox 1, the successor to Netscape Navigator, showing the same history in two different views.

- There are different temporal groupings possible; to my knowledge, the optimal size of the temporal chunks is unknown. For example, days could be used — as in this evaluation — or a mixture of days and weeks could be implemented in the same manner as Microsoft Internet Explorer 5. Alternatively, the temporal chunks could follow linguistic norms, such as morning, afternoon and evening (Larsen et al., 1999).

- Not all document types were tested. While I see no reason that the time to retrieve Web pages from history lists should differ from the retrieval of other document types, such as word-processor files or plain-text documents, the retrieval times may be different for different types of document.

- Finally, the participants had to rely on the cue for all supplied information, as they had no prior knowledge of when the document was viewed. The performance of the interfaces when the user is browsing

his or her own history is currently unknown.

Also unknown is the best way to present prior versions of a document to the user. This is studied in Chapter 5, before the guidelines presented in this chapter, and this thesis, are used to create a system that combines the retrieval of documents and versions (Chapter 6).

Chapter V

An Evaluation of Undo

In this chapter I compare four systems that support linear- and branched error-recovery (Figure 5.1). I show that, in a text editor, there is sufficient *mechanical* reason that forward error-correction (such as deleting erroneous text) should be favoured over version-retrieval systems (such as undo) for small error-recovery tasks, but larger errors are more quickly and easily corrected using version-retrieval systems.

The primary aim of the research presented in this chapter is to establish which error-recovery system is the best at retrieving versions that were created in the recent past. The design of this experiment is informed by the guidelines presented in the other sections that deal with versions: the study of user's tasks with versions (Section 2.4) and the study of existing version organisation systems (Section 3.3). This chapter will be used to inform the design of the system that combines document and version retrieval in the same interface, which is discussed in Chapter 6.

I begin this chapter by using keystroke-level analysis to determine the theoretical performance of an expert recovering from errors made while using a text-editor (Section 5.1). Then I present an empirical evaluation of error-recovery systems (Section 5.2) to determine how the editing environment affects error correction, and gather the participants' opinions on different error-recovery methods. This chapter is concluded with guidelines drawn from the two studies (Section 5.3).

5.1 Keystroke-Level Analysis of Error Recovery

When the user makes an error, or a series of errors, he or she must recover from it. There are many types of error-recovery mechanism, but in this section I will examine four. First, forward error-correction will be examined

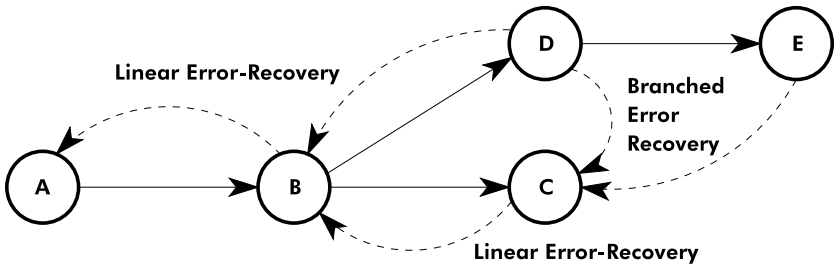


Figure 5.1: The states created in an editing task. State A is the initial state that is edited to create State B. This is edited to create State C, but this editing is abandoned (linear error recovery) to return to State B. State B is edited again to create State D. Finally, this state is edited to create State E. Branched error-recovery is returning to State C from D or E.

in Section 5.1.1. Next two types of undo — stack-based undo (Section 5.1.2) and history undo (Section 5.1.3) — will be discussed. Finally, tree-based error-recovery is examined in Section 5.1.4. These four methods were chosen because they exist in common applications, such as Microsoft Word in the case of stack-based undo, GNU Emacs in the case of history undo, and version control systems in the case of tree-based error-recovery. The error-recovery methods will be examined in the context of a text-editor as it is an interface that is familiar to most readers.

The examination in this section will use Keystroke-Level Analysis (KLA: Card et al., 1983). KLA is used to predict the time taken by an expert to perform a task by counting the number of mechanical actions required. It models a user that is an *expert* that does not make errors and performs the tasks without interruption. KLA does not take into account the cognitive difficulty of carrying out the task. As such, the times predicted by KLA may be faster than the measured results in an empirical evaluation. However, KLA gives us a base to compare interfaces before undertaking an empirical evaluation. In Section 5.1.5 I will use KLA to predict the time taken to perform two error-recovery tasks using the four different error-recovery mechanisms.

The notation used in the following sections is based on that used by Card et al. (1983), with two minor differences. First, arithmetic operators are explicitly used. Second, I use subscript text to clarify which button is pressed, such as $\mathbf{K}_{\text{Backspace}}$ for the `Backspace` key.

I will assume the text-editor uses the keyboard-navigation keys (`Home`, `End`, `Up`...) to move the text-entry point, that undo is invoked by pressing the `Control` and `z` keys simultaneously, that text can be selected by holding the `Shift` key while moving the text-entry point, and `Backspace` deletes the selected text or the character to the left of the text-entry point if no text is selected.

5.1.1 Forward Error-Correction

In this section I will perform a keystroke-level analysis of error-recovery using forward error-correction. I begin by looking at what forward error-correction is, before analysing the actions required to carry out linear and branched error-recovery tasks in a text-editor using a keyboard and mouse.

Forward error-correction occurs when creation and deletion actions are used to remove an error, rather than retrieving prior document states (Abowd and Dix, 1995). For example, using the `Backspace` or `Delete` keys to erase text in a word processor, or the eraser tool to remove unwanted lines from a image in a drawing program (linear error-recovery). Dix et al. (1997) claim that, unlike undo, forward error-correction does not require the user to switch from creating a document — working towards a goal — to thinking about a document and the past actions that were used to create the document. However, when using forward error-correction the user has to formulate the error correction actions, which are simple for basic text-editing tasks but could be complex. For example, recovering from a mail-merge using forward error-correction would be slower than using undo, as many more actions would be required.

How forward error-correction is used depends on the user. Consider a text-editing task where the user has to remove a sentence. A naïve user may repeatedly press the `Backspace` key to remove each character in turn, a more experienced user may press and hold `Backspace`, or an expert user may realise that `Backspace` deletes all selected text so he or she can select all erroneous text using the mouse or keyboard before pressing `Backspace`. Table 5.1 summarises the number of actions an expert-user can use to carry out linear and branched recovery tasks using two variants of forward error-

Forward Error-Correction (Keyboard)	
Linear	$\mathbf{M} + \mathbf{K}_{\text{Control-Shift-Left}} \times w_U + \mathbf{K}_{\text{Backspace}}$
Branched	$\mathbf{M} + \mathbf{K}_{\text{Control-Shift-Left}} \times w_U + \mathbf{M} + \mathbf{K} \times m_F$
Forward Error-Correction (Mouse)	
Linear	$\mathbf{M} + \mathbf{H}_{\text{Mouse}} + \mathbf{P}_{\text{Start}} + \mathbf{K}_{\text{Left}} + \mathbf{P}_{\text{End}} + \mathbf{K}_{\text{Backspace}}$
Branched	$\mathbf{M} + \mathbf{H}_{\text{Mouse}} + \mathbf{P}_{\text{Start}} + \mathbf{K}_{\text{Left}} + \mathbf{P}_{\text{End}} + \mathbf{M} + \mathbf{K} \times m_F$

M: Mental preparation time (1.35s); **K**: Key-press time (40WPM, 0.28s); **P**: Point to target with mouse (1.10s); **H**: Home hands on device (0.4s); w_U: Words to remove; m_F: Characters to retype.

Table 5.1: Keystroke-level analysis of two error recovery tasks using forward error-correction.

correction.

Interestingly, forward error-correction is the only method that I am aware of that would allow the user to recover from an error of *omission*, where content has been left out of the document. This is because there exists no prior state where the document had the omitted content, so the user cannot use version-retrieval to correct the error.

Forward Error-Correction Using the Keyboard To carry out a linear error-recovery task using the keyboard and forward error-correction requires the user to select each word by holding down the `Control`, `Shift` and `Left` keys simultaneously ($\mathbf{K}_{\text{Control-Shift-Left}} \times w_U$). All the selected text is then deleted by pressing `Backspace` ($\mathbf{K}_{\text{Backspace}}$). Fewer actions would be needed to complete linear error-recovery task if entire lines needed to be deleted, as the `Up` key could be used to select the lines to be deleted ($\mathbf{M} + \mathbf{K}_{\text{Shift-Up}} \times l_U + \mathbf{K}_{\text{Backspace}}$, where l_U is the number of lines to delete).

Using forward error-correction to carry out a branched error-recovery task requires more actions than carrying out a linear error-recovery task. The number of actions is equivalent to recovering from some text that was erroneously typed ($\mathbf{M} + \mathbf{K}_{\text{Control-Shift-Left}} \times w_U$), and recovering from text that was erroneously deleted by retyping the text ($\mathbf{M} + \mathbf{K} \times m_F$). The user does not have to press the `Backspace` or `Delete` keys to remove the erroneous text as typing into a text editor when text is selected automatically deletes the selected text.

Forward Error-Correction Using the Mouse For linear error-recovery, first the user has to home his or her hands on the mouse ($\mathbf{H}_{\text{Mouse}}$). The user then points to where the selection will start, presses the left mouse-button, and points to the end of the text to be deleted ($\mathbf{P}_{\text{Start}} + \mathbf{K}_{\text{Left}} + \mathbf{P}_{\text{End}}$). When the text is selected the user presses the `Backspace` key ($\mathbf{K}_{\text{Backspace}}$) to delete the text. The user does not have to home on the keyboard after the text has been selected with the mouse, as I assume the user's non-dominant hand is used to press the `Backspace` key.

Branched error-recovery using the forward error-correction method would first require the user to select the erroneous text ($\mathbf{M}+\mathbf{H}_{\text{Mouse}}+\mathbf{P}_{\text{Start}}+\mathbf{K}_{\text{Left}}+\mathbf{P}_{\text{End}}$) and then type the correct text ($\mathbf{M}+\mathbf{K}\times m_F$). The user does not have to press the `Backspace` or `Delete` keys to remove the erroneous text as typing into a text editor when text is selected automatically deletes the selected text.

5.1.2 Stack-Based Undo

Stack-based undo was examined in Section 3.3.1.1, where it was shown that stack-based undo alone cannot be used to complete the branched error-recovery task. In this section I will perform a keystroke-level analysis of error-correction using stack-based undo. I begin by analysing the number of actions required to carry out linear and branched error-recovery tasks in a text-editor using a keyboard, and then examine the actions required to complete the same tasks using a mouse.

Table 5.2 summarises the number of actions required to carry out linear and branched error-recovery using stack-based undo and a stack-based undo visualisation like that found in Adobe Photoshop (Figure 3.12(a)). The visualisation in Photoshop presents the user's actions in a list, with the most recently performed action at the bottom. When the user clicks on an action, the version of the document that was created just after the action had been carried out is retrieved (Adobe, 2002).

Using Stack-Based Undo with a Keyboard To recover from a linear error using stack-based undo the user has to to press `Control-z` to undo the prior actions until all the text has been removed ($\mathbf{K}_{\text{Control-z}} \times a_U$). The number

Stack-Based Undo	
Linear	$\mathbf{M} + \mathbf{K}_{\text{Control-z}} \times a_U$
Branched	$\mathbf{M} + \mathbf{K}_{\text{Control-z}} \times a_U + \mathbf{M} + \mathbf{K} \times m_F$
Stack-Based Undo Visualisation	
Linear	$\mathbf{M} + \mathbf{H}_{\text{Mouse}} + \mathbf{P}_{\text{State}} + \mathbf{K}_{\text{Left}}$
Branched	$\mathbf{M} + \mathbf{H}_{\text{Mouse}} + \mathbf{P}_{\text{State}} + \mathbf{K}_{\text{Left}} + \mathbf{M} + \mathbf{K} \times m_F$

M: Mental preparation time (1.35s); **K**: Key-press time (40WPM, 0.28s); **P**: Time to point to target with mouse (1.10s); **H**: Time to home hands on device (0.4s); a_U: Undo actions to perform; m_F: Characters to retype.

Table 5.2: Keystroke-level analysis of two error recovery tasks using stack-based undo.

of states recorded can be quite different between applications, so the number of actions required to revert a document to a prior state (a_U) depends on the system as well as the number of actions the user has performed.

Branched error-recovery is more difficult than linear error-recovery, as only the current branch is retained: the user has to retype the characters that were erroneously deleted ($\mathbf{M} + \mathbf{K} \times m_F$) after invoking undo ($\mathbf{M} + \mathbf{K}_{\text{Control-z}} \times a_U$).

Using an Undo-Visualisation To perform linear error-recovery using an undo visualisation first requires the user to home on the mouse ($\mathbf{H}_{\text{Mouse}}$). Then the correct state is selected by pointing to the state and pressing the left mouse-button ($\mathbf{P}_{\text{State}} + \mathbf{K}_{\text{Left}}$).

Branched error-recovery still requires the user to retype the characters that were erroneously deleted ($\mathbf{M} + \mathbf{K} \times m_F$) after performing the undo as only the current branch is stored on the stack.

5.1.3 History Undo

While not commonly found outside the Emacs family of text editors (Harvey, 2003), history undo is worth examining as it similar to both stack-based undo and tree-based error-recovery. It was examined in detail in Section 3.3.1.1, where it was shown that history-undo alone can be used to complete both linear and branched error-recovery. While Emacs does not provide a visuali-

History Undo	
Linear	$\mathbf{M} + \mathbf{K}_{\text{Control-z}} \times a_U$
Branched	$\mathbf{M} + \mathbf{K}_{\text{Control-z}} \times (a_U + a_F)$
History Undo Visualisation	
Linear	$\mathbf{M} + \mathbf{H}_{\text{Mouse}} + \mathbf{P}_{\text{State}} + \mathbf{K}_{\text{Left}}$
Branched	$\mathbf{M} + \mathbf{H}_{\text{Mouse}} + \mathbf{P}_{\text{State}} + \mathbf{K}_{\text{Left}}$

M: Mental preparation time (1.35s); **K**: Key-press time (40WPM, 0.28s); **P**: Point to target with mouse (1.10s); **H**: Home hands on device (0.4s); a_U: Undo actions to remove text; a_F: Undo actions to recreate text.

Table 5.3: Keystroke-level analysis of error-recovery tasks using history undo.

sation of history-undo, it is possible to envision one similar to that provided for the stack-based undo system found in Adobe Photoshop (Figure 3.12(a)). In this section I will perform a keystroke-level analysis of error-recovery tasks in a text-editor using history-undo with both a keyboard and mouse.

Table 5.3 summarises the number of actions required to recover from an error using history undo.

Using History-Undo with a Keyboard To recover from a linear-error using history-undo with a keyboard requires the same number of actions as performing the task using stack-based undo with a keyboard ($\mathbf{M} + \mathbf{K}_{\text{Control-z}} \times a_U$). However, history undo allows branched error-recovery to be completed in potentially fewer actions than stack-based undo as the user does not have to retype the older branch ($\mathbf{M} + \mathbf{K} \times m_F$), rather history undo can be used to undo the a_F deleting actions ($\mathbf{M} + \mathbf{K}_{\text{Control-z}} \times (a_U + a_F)$).

Using a History-Undo Visualisation To complete linear error-recovery the user must select the past state, in the same way as is done with the stack-based undo visualisation (Section 5.1.2). However, unlike the stack-based undo visualisation, fewer actions are required to complete the branched error recovery task, as the prior state will still be able to be selected ($\mathbf{M} + \mathbf{H}_{\text{Mouse}} + \mathbf{P}_{\text{State}} + \mathbf{K}_{\text{Left}}$). This time, of 3.13s, should be the same for editors outside the text-editing domain.

Tree-Based Error Recovery	
Linear	$\mathbf{M} + \mathbf{H}_{\text{Mouse}} + \mathbf{P}_{\text{State}} + \mathbf{K}_{\text{Left}}$
Branched	$\mathbf{M} + \mathbf{H}_{\text{Mouse}} + \mathbf{P}_{\text{State}} + \mathbf{K}_{\text{Left}}$

M: Mental preparation time (1.35s); **H**: Home hands on device (0.4s); **P**: Point to target with mouse (1.10s); **K**: Key-press time (40WPM, 0.28s).

Table 5.4: Keystroke-level analysis of error-recovery tasks using tree-based error recovery.

5.1.4 Tree-Based Error Recovery

Tree-based error recovery systems are commonly found as part of version-control systems (Section 3.3.2.1). However, an undo mechanism could be implemented that used a tree, rather than a stack or list, as the main data structure. For example, the Visage data-visualisation system (Derthick and Roth, 2000) provides a tree-based undo mechanism to allow users to return to past states. In this section I will perform a keystroke-level analysis of error-recovery using tree-based error recovery. I will analysing the number of actions required to carry out linear and branched error-recovery tasks in a text-editor.

Version-control systems differ dramatically from each other in how commands are issued. The commands are more complex than the simple undo and redo commands as the user has to be able to select different branches. Therefore, rather than basing the theoretical tree-based error recovery system on an existing program, I will assume that its basic operation is similar to the other two undo visualisation techniques when using a mouse.

Using a Tree-Based Error Recovery Visualisation A tree-based error recovery visualisation requires the user to perform the same number of actions to complete either the linear or branched error-recovery tasks: select the state after homing on the mouse (Table 5.4). This is the same number of actions that is required by the history-undo visualisation. Like the history-undo visualisation, the time taken to carry out error-recovery using a tree visualisation should be the same for interfaces outside the text-editing domain.

5.1.5 Comparison of Recovery Methods

To make comparison between the different recovery methods easier, we created a text-editing scenario where an author is writing a passage from Carroll (1997). The author starts with a blank document (State A) and types the following into the editor.

> "Cheshire-Puss," she began, "would you tell me, please, which way I ought to go from here?"
>
> "That depends a good deal on where you want to get to," said the Cat.

This generates 162 characters, 3 lines and 31 words, creating State B. Editing continues with the following 97 characters, 2 lines and 18 words being generated to create State C.

> "I don't care much where–" said Alice.
>
> "Then it doesn't matter which way you go," said the Cat.

The user then decides to revert to State B (linear error-recovery) before typing the following into the editor to create State D.

> "I want to see the Doormouse and the White Rabbit," exclaimed Alice.
>
> "Enjoy the tea," said the Cat.

Finally, State D is reverted to State C (branched error-recovery).

Table 5.5 summarises the time taken to carry out the error-recovery tasks in the above scenario using the error-recovery systems discussed in the previous sections, using the statistics shown in Figure 5.2. Forward error-correction has the shortest estimated time for the linear error-recovery task at 2.19s. However, there are only two lines to delete in the example text ($l_U = 2$), so longer passages of text would take more time to delete (0.28s per line). Assuming that each word is considered an action by the system ($a_U = w_U$), then stack-based undo and history-undo both take the longest time to complete the linear error-recovery task due to the many times `Control-z` must be pressed ($a_U = 18$). However, the three visualisation methods (stack-based undo visualisation, history undo visualisation,

	Task	
Recovery Method	**Linear**	**Branched**
Forward Error-Correction (Keyboard)	2.19s	30.42s
Forward Error-Correction (Mouse)	4.51s	33.02s
Stack-Based Undo	6.39s	34.90s
Stack-Based Undo Visualisation	3.13s	31.36s
History Undo	6.39s	11.43s
History Undo Visualisation	3.13s	3.13s
Tree-Based Error Recovery	3.13s	3.13s

Table 5.5: Estimated times to complete two error-recovery tasks using different error-recovery methods. The statistics used to calculate the times were taken from Card et al. (1983).

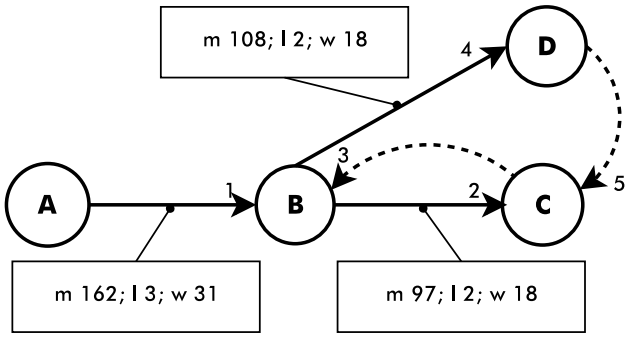


Figure 5.2: Example statistics used to generate the times in Table 5.5. (m the number of characters; l the number of lines; w the number of words.)

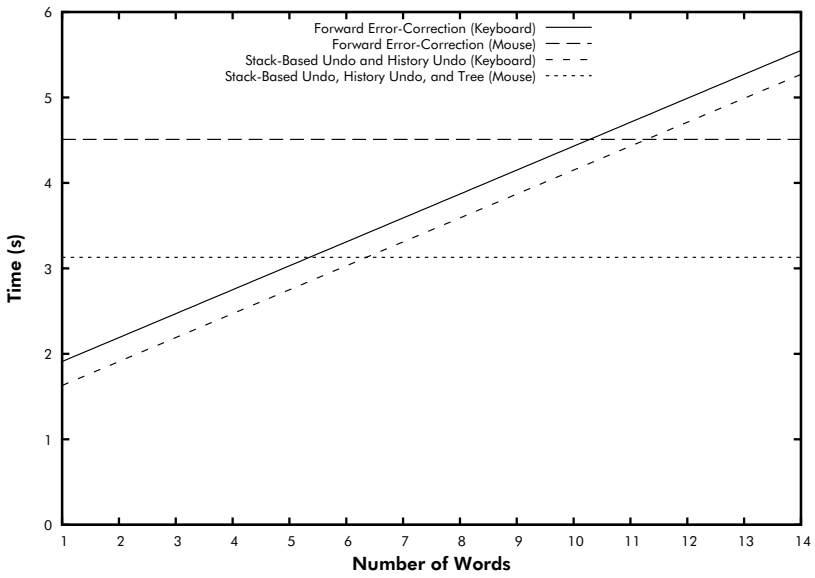


Figure 5.3: Plot of the estimated time taken for an expert user to complete linear error-recovery using different methods, assuming the number of actions to undo equals the number of words to undo.

and tree-based undo) require the same amount of time to complete the linear error-recovery task (3.13s), because all three methods require the user to perform the same sequence of actions: $\mathbf{M} + \mathbf{H}_{\text{Mouse}} + \mathbf{P}_{\text{State}} + \mathbf{K}_{\text{Left}}$.

If we assume that the number of actions to undo is always equal to the number of words to remove ($a_U = w_U$), then we can examine the time taken to complete linear error-recovery in the general case (Figure 5.3). An expert user should be able to carry out a linear error-recovery task in the least time using stack-based undo and history undo, if the number of words to be undone was six words or fewer. Beyond this point the stack-based undo visualisation, history undo visualisation and tree become the quickest methods of recovery, as they take constant time (3.13s). Forward error-correction using a keyboard takes an extra keystroke (`Backspace`) compared to the keyboard-based undo methods, so it remains consistently slower. Recovering from an error using forward error-correction with a mouse takes constant time (4.51s), but error-recovery is consistently slower than that performed using the undo visualisation methods with a mouse. This is because forward error-correction with a mouse requires the user to point to two locations (the

start and end of the text to be removed) rather than one, and requires an extra keystroke (`Backspace`).

The time to retype the text in the branched error-recovery task ($\mathbf{M}+\mathbf{K}\times m_F$) makes the forward error-correction and stack-based undo error-recovery methods slower than the other methods, when carrying out the linear error-recovery task. However, using the history-undo visualisation and tree-based error-recovery methods to complete the branched error-recovery task should take the same amount of time as that required to complete the linear error-recovery task.

For a trivial error, forward error-correction using backspace would be the fastest method, alongside stack-based undo and history undo, taking 0.28s for a single-character error, and 0.56s for a branched error of two characters (assuming stack-based undo and history undo will allow the typing of a single character to be undone).

No error recovery method is the fastest at both linear *and* branched error recovery tasks. However, there is compelling evidence from the keystroke-level analysis to suggest that either history undo using a visualisation, or tree-based error-correction, which also uses a visualisation, would be fast at branched error-recovery.

5.2 Empirical Evaluation of Recovery Methods

An empirical evaluation was undertaken to determine which error-recovery method people prefer, to see if users' preferences are the same across editing domains, and to validate the performance predictions of the keystroke-level analysis (Section 5.1). KLA may be inaccurate because it assumes that the cognitive difficulty of the different systems is the same, and that the user carries out all the tasks without error. The empirical evaluation may also help determine if providing a visualisation of undo allows for faster error-recovery. Finally, the raw data that is collected allows us to analyse the typing patterns and drawing behaviour of participants, which will allow us to create more effective error-recovery methods.

An overview of the evaluation is presented in Section 5.2.1 and the design of the two experiments that make up the evaluation is given in Section 5.2.2.

Next the different document-editing interfaces are introduced (Section 5.2.3), before the error-recovery methods are discussed in Section 5.2.4. The participant details are examined, before the experimental procedure is then introduced (Sections 5.2.5 and 5.2.6). The experimental concerns are discussed in Section 5.2.8, before the results and discussion are presented (Sections 5.2.7 and 5.2.9).

5.2.1 Overview

The empirical evaluation consisted of two experiments. The first experiment tested four error-recovery methods — forward error-correction, undo, undo visualisation, and tree-based undo (Section 5.2.4) — in the context of a text-editor (Section 5.2.3.1). The second tested the same error-recovery methods in the context of a drawing program (Section 5.2.3.2). Both experiments required the participant to create a document, perform a linear error-recovery task, create an alternate version, and carry out a branched error-recovery task to return to the original version.

The history-undo method (Section 5.1.3) could have been tested. However, it is not widely used outside the Emacs family of editors, and it has a similar expressive power to the tree-based error-recovery method, including the same theoretical task-completion times for the visualisation, as can be seen in Table 5.5. Therefore it was decided not to test the history-undo method.

5.2.2 Design

The text editor and drawing program experiments had the same design: a 4×2 repeated measures analysis of variance (ANOVA), with time as the dependant variable. The factor 'error-recovery method' had four levels — forward error-correction, undo, undo visualisation, and tree — while the second factor, error-recovery task, had two levels — linear and branched.

5.2.3 Interfaces

Each experiment used a different interface: a text editor (Section 5.2.3.1) and a drawing program (Section 5.2.3.2). Two different editors were used to see if

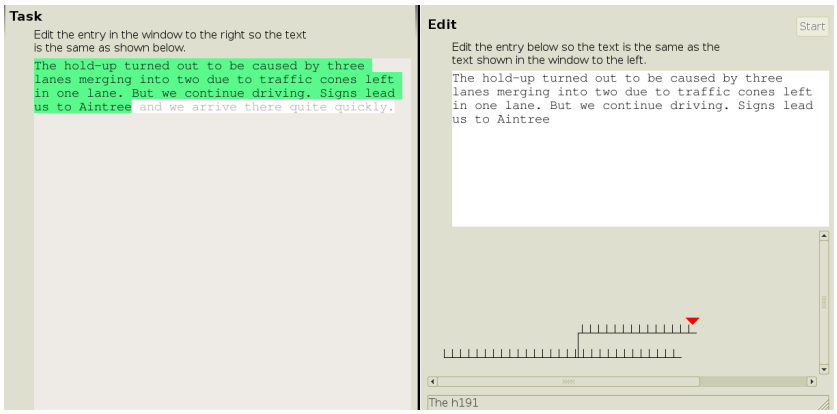


Figure 5.4: The text-editor interface. The task cue is to the left, with the correct text highlighted green (up to and including 'Aintree'). At the bottom of the edit window is the visualisation for the tree-based error-recovery method.

the subjective measures were the same across two different editing domains, and whether the time taken to recover from errors using the visualisations were constant across both domains, as is predicted by the keystroke-level analysis of the text editor. The text-editing and drawing domains were chosen as they are quite different from each other, but they were still familiar to most participants, unlike the sound-editing or movie-editing domains for example.

Both experiments used the same apparatus, which is detailed in Section 5.2.3.3.

5.2.3.1 Text Editor Interface

The text editor interface consists of two windows (Figure 5.4). The left-hand window shows the task to be completed, while the right-hand window allows the user to enter and edit text. If the error-recovery method had a visualisation, it appears below the text-entry area in the Edit window. Different windows were used for the task display and text entry. This allowed the participant to see what text should be typed *and* what text had been entered. It was necessary to show the participant both so the error could be quickly seen and corrected.

The keyboard-navigation keys move the text-entry point, text is selected by holding `Shift` while navigating, and undo and redo are bound to the

`Control-z` and `Control-y` key sequences respectively. The text-display and text-entry areas are 50 characters wide, and line-breaking is performed automatically. The text itself is displayed using the Free Mono typeface (Peterlin, 2002) rendered using anti-aliasing at 14 points.

When erroneous text is entered three things happen: the characters that were mistyped are highlighted red in the Edit window, the characters that should have been typed are highlighted red in the Task window, and the machine emits a beep. Correctly entered text is highlighted green in the Task window, so there is a visual distinction between the Task and Edit windows — as the error-recovery method did not always have a visualisation to provide a distinction.

The system automatically detects that the task had been completed. When the text has been left in the completed state for 500ms the text-entry is made unavailable ('greyed-out') and a beep is emitted. The delay forced the participants to deliberately leave the system in a finished state, rather than completing an error-recovery task by pressing the `Backspace` or undo key and waiting for the system to respond. The delay of 500ms was chosen as this is the maximum delay for feedback according to Miller (1968).

Software logged all user-actions and cued all tasks.

5.2.3.2 Drawing Editor Interface

Like the text editor, the drawing interface has two windows (Figure 5.5). The left-hand window showed the drawing that is copied by the participant, while the right-hand window contains a simple drawing program. The sample image is a black and white line-drawing with the name of the image above it. Lines in the sample image are 2 pixels wide and were anti-aliased.

The drawing program has two tools: a freehand drawing tool (a 'pencil') and an eraser. Visualisations appear below the drawing canvas in the Edit window, if the error-recovery interface requires it. The canvas is 300 pixels wide by 400 pixels high. Lines drawn by the pencil are black, 3 pixels wide and anti-aliased. The line width is wider than the lines used to draw the sample images so the participants are less inclined to attempt to make an exact copy of the sample image.

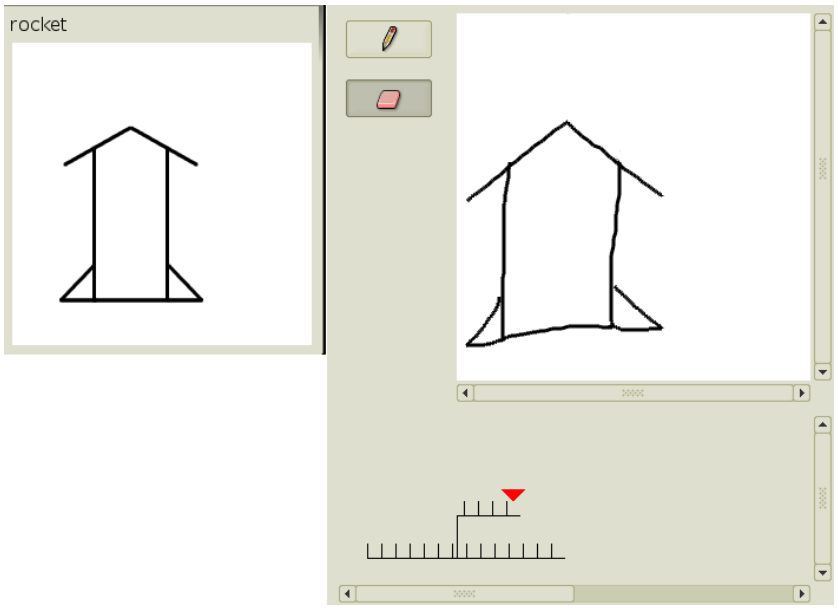


Figure 5.5: Drawing-editor interface. The task cue is to the left, with as short textual description of the picture above it. At the bottom of the edit window is the visualisation for the tree-based error-recovery method.

The participant only had to create an approximate version of the sample drawing because I was interested in how people perform error-recovery when the errors are not precise, unlike typing. A stopwatch was used for timing: the participant indicated when he or she wanted to start drawing, and timing was finished when his or her hand was moved from the mouse.

5.2.3.3 Apparatus

Both interfaces were displayed on a full-colour 1600×1200 pixel 48cm monitor (112dpi) that had a 75Hz refresh rate and a dot-pitch of 0.26mm diagonal and 0.22mm horizontal. The computer was an AMD 1800+ Athlon CPU and 1GB of physical RAM. It ran a Linux (version 2.6), used XFree86 (version 4.4) as the windowing system and GTK+ 2.2 was used as the widget set for both interfaces. Both the keyboard and mouse were connected to standard PS/2 ports.

5.2.4 Error-Recovery Methods

There were four error-recovery methods tested in the evaluation: forward error-correction, stack-based undo, stack-based undo visualisation, and the tree-based undo visualisation.

The forward error-correction method (Section 5.1.1) allows forward error-correction to be carried out, but not undo. In the case of the text-editor this allows the participant to use any of the text-navigation, typing and deleting keys. For the drawing program this allows only the the pencil and eraser tools to be used.

The stack-based undo method is the same as the forward error-correction method except the participant can undo editing by typing `Control-z` or redo by typing `Control-y` (Section 5.1.2). So the participants are constrained, there is no visualisation of the undo stack, and there are no toolbar buttons that corresponded to undo or redo. In the text-editor interface, undo states are created after the user pauses for 200ms; in the drawing interface states are created after the user releases the mouse button, indicating that a stroke has been finished.

The stack-based undo visualisation error-recovery method (Section 5.1.2) is the same as the stack-based undo method except a visualisation of the undo stack is provided. The visualisation appears below the editing area in both the text-editing and drawing interfaces (Section 5.2.3). It consists of a timeline — which grows from left to right — with small black tics added to the timeline as undo states are created. There are 6 pixels separating each tic, each of which is 6 pixels high and 1 pixel wide. A red marker indicates the current location on the timeline, which is normally at the end, 3 pixels after the most recently created tic. To undo an action, the mouse cursor is used to drag the red marker to a point just before the erroneous action had occurred. So there is no ambiguity about the current state of the document, the red marker only points to a position 3 pixels after a tic mark. When the user drags the pointer using the mouse, the pointer 'jumps' to valid positions and the text or drawing is immediately updated. When editing resumed after an undo, the points to the right of the red marker are removed. The behaviour of the stack-based undo visualisation is different to

Stack-Based Undo Visualisation	
Linear	$\mathbf{M} + \mathbf{H}_{\text{Mouse}} + \mathbf{P}_{\text{Marker}} + \mathbf{K}_{\text{Left}} + \mathbf{P}_{\text{State}}$
Branched	$\mathbf{M} + \mathbf{H}_{\text{Mouse}} + \mathbf{P}_{\text{Marker}} + \mathbf{K}_{\text{Left}} + \mathbf{P}_{\text{State}} + \mathbf{M} + \mathbf{K} \times m_F$
Tree-Based Undo Visualisation	
Linear	$\mathbf{M} + \mathbf{H}_{\text{Mouse}} + \mathbf{P}_{\text{Marker}} + \mathbf{K}_{\text{Left}} + \mathbf{P}_{\text{State}}$
Branched	$\mathbf{M} + \mathbf{H}_{\text{Mouse}} + \mathbf{P}_{\text{Marker}} + \mathbf{K}_{\text{Left}} + \mathbf{P}_{\text{State}}$

M: Mental preparation time (1.35s); **H**: Home hands on device (0.4s); **P**: Point to target with mouse (1.10s); **K**: Key-press time (40WPM, 0.28s). m_F: Characters to retype.

Table 5.6: Keystroke-level analysis of the two error-recovery methods that used visualisations. The above formulae assumes that the user clicks a marker and drags it to the correct state, rather than simply clicking on the correct state, as was assumed in Tables 5.2 and 5.4

that described in Section 5.1.2: preliminary trials showed that the current behaviour was easier to use, despite it being suboptimal. Table 5.6 presents the keystroke-level analysis of the error-recovery method.

The tree-based undo visualisation error-recovery method (Section 5.1.4) is similar to the stack-based undo visualisation method in that a visualisation is provided. However, a branch is created when the user commences editing after performing an undo, instead of the points to the right of the marker disappearing. The behaviour of the tree-based undo visualisation error-recovery method is different to that described in Section 5.1.4 for the same reasons as described for the stack-based undo visualisation. Table 5.6 presents the keystroke-level analysis for the tree-based undo visualisation. Figures 5.4 and 5.5 show the visualisation for the tree.

5.2.5 Participant Details

33 volunteer students participated in the study, with 21 using the text-editor and eleven using the drawing program. All participants were rewarded with a $5 shopping voucher.

5.2.6 Procedure

The two experiments, though using different document editors, used a similar procedure to test linear and branched error-recovery. First, the experiment

was introduced to the participant by explaining the reason for the experiment, and by asking some background questions.

To mitigate learning-effects, the order that participants tested the error-recovery methods was balanced. The error-recovery methods that are based on undo would be more difficult to use if the participant did not know the past document states — so, in both the drawing and typing experiments the participant had to create the document before he or she could correct the errors, regardless of the error-recovery method that was used.

The error-recovery method was introduced by requiring the user to perform a training task, which was the same for all error-recovery methods using a particular editing interface. The training task, like all editing tasks, consisted of creating a document and performing a series of modifications to that document that followed the same pattern as the tasks examined in Section 5.1.5. Figure 5.6 shows the states for the training task using the drawing program. For the text editor, the training task started with a blank document (State A) and required the participant to carry out the following subtasks.

Typing Task 1 Type `I␣am␣a␣fish.` (creating State B).

Typing Task 2 Add `␣I␣like␣to␣swim␣in␣the␣sea.` to the end of the previous text (creating State C).

Linear Error-Recovery Remove the last sentence (linear error-recovery, returning to State B).

Typing Task 3 Add `␣Eating␣plankton␣is␣fun.` to the end of the current text (creating State D).

Branched Error-Recovery Correct the text so the last sentence is replaced with `␣I␣like␣to␣swim␣in␣the␣sea.` (branched error-recovery, returning to State C).

The participant undertook the timed task with a particular error-recovery method after the training task had been completed. There would be a learning effect if the participant carried out the same set of tasks more than once,

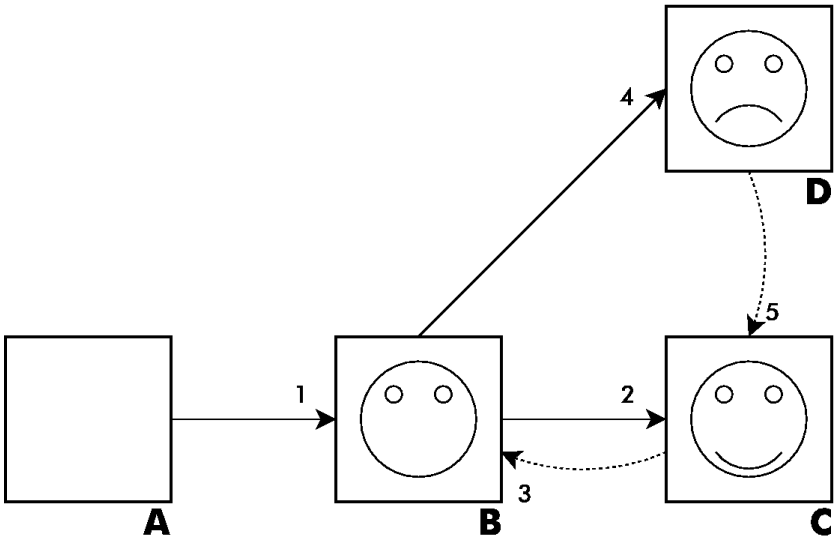
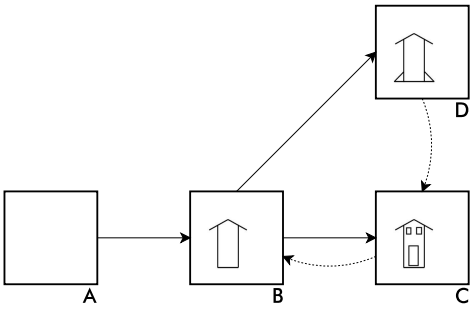
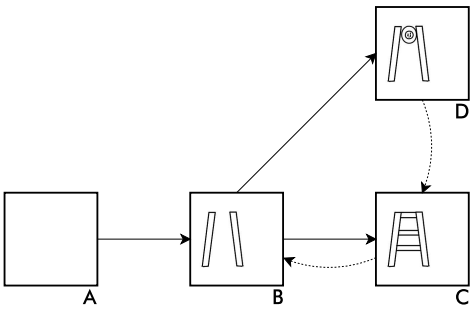


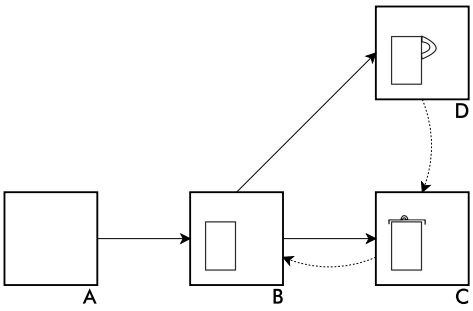
Figure 5.6: The different states for the training task using the drawing interface. Participants started with a blank canvas (State A) before drawing a face without a mouth (State B). A smile was added (State C), which was then altered during a linear error-recovery task to recreate the face without a mouth (State B). A frown was then added to the face (State D). The final error-recovery subtask was to correct the document so it was in State C again, a branched error-recovery task.



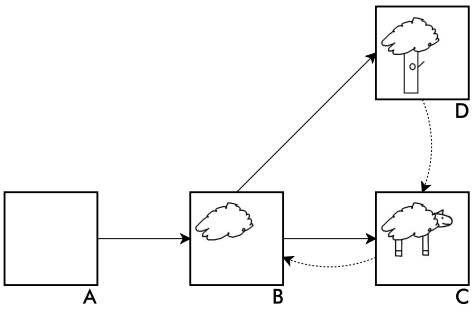
(a) A house (State C) and a rocket (State D).



(b) A ladder (State C) and sushi (State D).



(c) A rubbish bin (State C) and a mug (State D).



(d) A sheep (State C) and a tree (State D).

Figure 5.7: The timed drawing tasks. All tasks followed the same pattern as the training task (Figure 5.6) by creating two different documents based on a common base.

so there were four different drawing tasks and four different typing tasks, which were randomly assigned to a error-correction method before each participant started the experiment. The timed drawing-tasks are shown in Figure 5.7, and they follow the same pattern as the training task (Figure 5.6). Like the training task, the timed text-editing tasks followed the same pattern as the drawing tasks. The document would start out blank (State A) and a sentence would be added to get into State B. Another sentence would be added to get into State C, which would then be reverted to State B (linear error-recovery). The final sentence would then be typed to create State D before returning to State C (branched error-recovery).

Table 5.7 shows the mean number of characters and words that the participants had to type to create the states. There were fewer characters to

Task	Words		Characters		Predicted Time			
	Range	$\overline{x}$	Range	$\overline{x}$	FEC	Undo	Undo V'	Tree
Typing 1	16–27	20.25	90–129	104	30.5s	30.5s	30.5s	30.5s
Typing 2	10–16	13.25	69–78	74	22.1s	22.1s	22.1s	22.1s
Linear ER					5.3s	5.1s	4.2s	4.2s
Typing 3	12–16	14.50	83–95	87	25.7s	25.7s	25.7s	25.7s
Branched ER					26.5s	27.5s	25.0s	4.2s

Table 5.7: The predicted time taken to move between various states using the different error-recovery methods (where FEC stands for 'forward error-correction', and Undo V' stands for 'undo visualisation'). The estimated times for the typing subtasks (A–B, B–C, and B–D) are based on the formula $\mathbf{M} + \mathbf{K} \times m$ where $\mathbf{M}$ is the mental preparation time (1.35s), $\mathbf{K}$ is the time to press a key (40 WPM, 0.28s) and m is the mean number of characters to type. The other times are based on the formulae in Tables 5.1, 5.2 and 5.6.

be typed to move between States B and C as they had to be typed twice for the forward error-correction, undo, and undo visualisation error-recovery methods, and I did not want to overly frustrate the participants. The text was taken from holiday-travel reports listed on a New Zealand newspaper Website (Fairfax New Zealand Limited, 2004). Travel reports were chosen because they had simple language and grammar. For example, the sample sentences had no dialogue and the participant did not have to explicitly create any line-breaks. When typing, the participant could use any available error-recovery method to correct errors. However, when performing the linear and branched error-recovery tasks, the participant was asked to used only the error-recovery method that was being tested.

In both experiments, the experiment supervisor explained what the state of the document should be *after* the participant had finished the subtask. This gave the participant time to prepare his or her actions, and should reduce the affect of the possible breakdown caused by undo (Dix et al., 1997). Because of this mental preparation time, any difference in speed between the error-correction methods should be due to the mechanical differences in the systems, not the cognitive differences.

For both the drawing and text-editing interfaces, the participant was asked if he or she agreed with the statement "It was easy to correct errors using this method", after completing the timed document editing task. The

responses were measured on a five-point Likert scale (5 was agree).

After using all four error-recovery methods in the text-editor experiment, the participant was then asked to perform a task to determine which error-recovery method would be used if not overdetermined by the interface. The participant was asked to describe the weather, pretending that he or she was writing an email to a friend. The participant was then asked to alter the message so it was more formal, such as part of a letter to the Vice Chancellor of the University. The final task was to alter the document so it was once again like an email to a friend. The interface presented the tree-based error-recovery method, but it was explained to the participant that he or she could use any method to complete the task.

In the drawing experiment the participant was asked if he or she would use the tree error-recovery method in the future if given the opportunity.

Finally, in both experiments the participant ranked the error-recovery methods according to the one he or she preferred.

5.2.7 Results

All participants completed the tasks, with either the text editor or drawing program, without difficulty. The results from the text-editor experiment are presented in Section 5.2.7.1, while the results from the experiment with the drawing program are presented in Section 5.2.7.2.

5.2.7.1 Text Editor Results

There was a significant difference in the mean time to complete the error-recovery subtasks using the different error-correction interfaces ($F(3, 60) = 52.9$, $p < 0.01$). The subtasks were completed in the shortest time using the tree error-recovery method, with a mean time of 5.7s ($\sigma = 4.8$), and the slowest using undo, with a mean time of 15.3s ($\sigma = 11.6$). Participants completed the error-recovery tasks in a mean time of 15.0s ($\sigma = 11.1$) using the undo visualisation, while forward error-correction took 11.4s ($\sigma = 9.0$). Post-hoc analysis shows there was a significant difference between the time taken to complete the subtasks using the tree interface, and the undo and undo-visualisation interfaces (Tukey Test, HSD = 9.1).

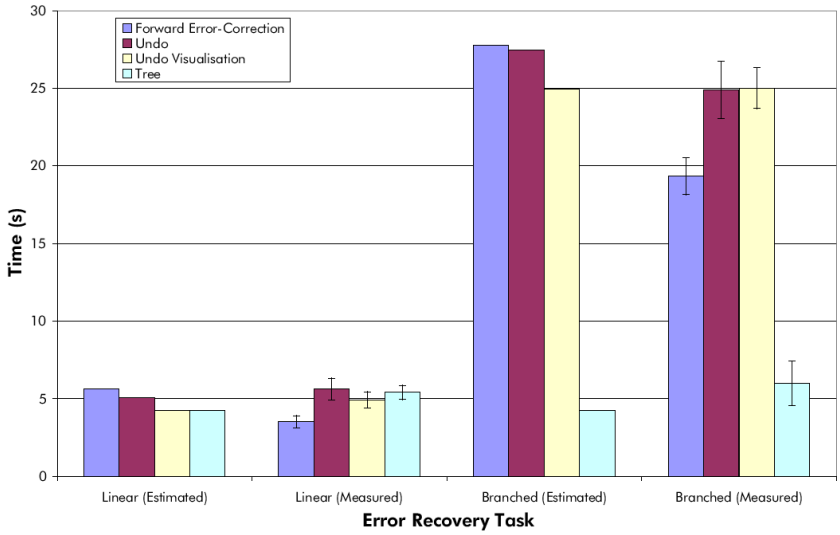


Figure 5.8: Estimated and measured times taken to complete the linear and branched error-recovery tasks using the different error-correction methods. The estimated times are from Table 5.7.

The participants completed the linear error-recovery subtask quicker than branched error-recovery, taking a mean time of 4.9s ($\sigma = 2.5$) to complete, compared to a mean time of 18.8s ($\sigma = 10.2$). There was a significant difference between the time taken to complete the error-recovery subtasks ($F(1, 20) = 277.8$, $p < 0.01$).

Figure 5.8 shows the mean times to complete the different text-editing tasks using the different error-correction interfaces. There was a significant interaction between the interface and error-recovery method independent variables ($F(3, 60) = 44.5$, $p < 0.01$). The reason for the interaction is twofold. First, the forward error-correction interface allowed participants to complete the linear error-recovery faster than the other three interfaces, with a mean time of 3.5s ($\sigma = 1.8$), compared to the slowest, undo, that took 5.6s ($\sigma = 3.2$). Second, participants completed the branched error-recovery task in less time using the tree interface, taking a mean time of 6.0s ($\sigma = 6.5$). The slowest interface at completing the branched error-recovery task was the undo visualisation, which took participants a mean time of 25.0s to complete ($\sigma = 6.1$). When carrying out the linear error-recovery subtask using the forward error correction method, significantly more participants used the

	Typing Task 1		Typing Task 2		Typing Task 3	
	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ
Typing	14.3	7.7	10.7	5.4	12.0	5.6
Deletion	2.3	2.7	2.3	2.2	2.7	2.0
Total	17.8	9.5	13.0	7.1	14.8	7.0

Table 5.8: The mean number of undo states, created by typing and deleting actions, during the three typing subtasks. The number of undo states created indicates how often `Control-z` has to be typed to recover from an error using the undo error-recovery method. It also indicates how far the pointer on the timeline had to be moved to recover from and error using the undo visualisation or tree error-recovery methods.

mouse to select text than the keyboard (18 compared to 3, $\chi^2 = 9.33$, DF $= 1$, $p < 0.01$).

For the open task, when asked to return to the email from the letter, the participants were able to choose the error-correction method used. They were more likely to use the visualisation, with 15 using the visualisation compared to 6. However, this is a marginal effect ($\chi^2 = 3.05$, DF $= 1$, $p = 0.08$).

The most commonly used text editor among the participants was XEmacs (which was the standard text editor in the Department of Computer Science), with 14 users, but all participants said that they commonly used an editor that provided the stack-based undo semantic (Section 3.3.1.1). However, undo was not used by one participant before this experiment. Version-Control systems were used by 11 of the 21 participants, but 6 of those said that they did not use the systems for error recovery, primarily using them to allow many users to share source-code.

The mean typing speed of the participants was between 39.9 and 46.3 words per minute (3.8–4.0 characters per second), depending on the task. The higher words-per-minute count was for the open typing task, which was carried out at the end of the experiment; the higher characters-per-second count was for the second typing task.

Table 5.8 lists the mean number of undo states created in each of the three initial typing subtasks. No participant used undo, or either of the visualisations, to correct errors in the typing tasks. Therefore the number of deleting states created (2.3–3.5) gives an indication as to the accuracy of the

participants typing.

Subjective Measures After using each error-recovery method, the participants responses to the statement "It was easy to correct errors using this method" was measured on a five-point Likert scale, with 5 being agree. The mean rating for forward error-correction, the favoured interface, was 4.1 ($\sigma = 0.8$) while undo, the least favoured interface, rated 3.3 ($\sigma = 1.0$). The rating for the undo visualisation error-recovery method was close to that of undo (3.4, $\sigma = 0.9$), but rating for tree was closer to forward error-correction (4.0, $\sigma = 0.8$). The mean response across all error-recovery methods was 3.7 ($\sigma = 0.9$), which was above the expected value of 3.55 ± 0.12, reported by Nielsen and Levy (1994).

However, at the end of the experiment, when asked to rank the interfaces, 11 of the 21 participants ranked the tree interface first, compared to the next highest, forward error correction, that 5 participants ranked as the best interface.

5.2.7.2 Drawing Program Results

There was a significant difference in the mean time to complete the error-recovery subtasks using the different error-correction interfaces ($F(3, 30) = 56.8$, $p < 0.01$). The error-recovery subtasks were completed in the shortest time using the tree error-recovery method, with a mean time of 4.4s ($\sigma = 1.9$), and the slowest using forward error-correction, with a mean time of 19.9s ($\sigma = 5.8$). Participants took 8.1s ($\sigma = 5.7$) and 10.0s ($\sigma = 6.0$) to complete the error-recovery subtasks using the undo and undo visualisation methods respectively. Post-hoc analysis shows there is a significant difference between the time taken to complete the subtasks using forward error-correction and the three other error-recovery methods (Tukey Test, HSD = 6.0).

Participants completed the linear error-recovery task in significantly less time than branched error-recovery ($F(1, 10) = 104.7$, $p < 0.01$), taking 7.8s ($\sigma = 7.1$) compared to 13.2s ($\sigma = 7.4$).

As can be seen in Figure 5.9, there was a significant interaction between the interface and subtask independent variables ($F(3, 30) = 14.4$, $p < 0.01$). There are two possible reasons that have been identified for this interaction.

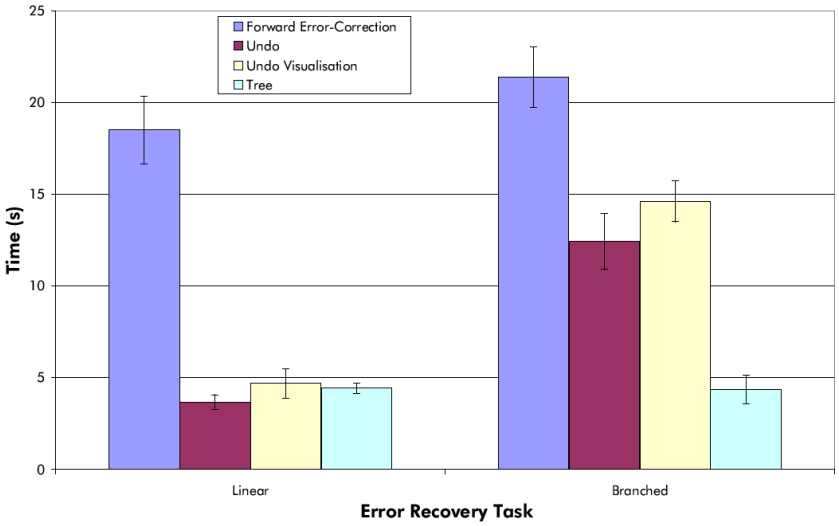


Figure 5.9: Measured times taken to complete the drawing tasks using the different error correction methods.

First, participants were slower at carrying out the linear error-recovery using forward error-correction compared to the other three error-recovery methods. The mean time it took participants to complete the linear error-recovery subtask using forward error-correction was 18.5s ($\sigma = 6.1$), compared with the fastest interface, undo, which took participants 3.7s ($\sigma = 1.3$). Second, participants were slower at carrying out branched error-recovery using forward error-correction compared to the other three methods, and participants were faster at carrying out this subtask using the tree interface. It took participants a mean time of 21.4s ($\sigma = 5.5$) to complete branched error-recovery using forward error-correction, compared to 4.4s ($\sigma = 2.6$) using the tree interface, and 12.4s ($\sigma = 5.0$) and 14.6s ($\sigma = 3.7$) for the undo and undo visualisation interfaces respectively.

While there was no accurate data kept on the number of undo nodes created in the drawing experiment, generally the number of nodes was far less than for the typing experiment. A node would be created for each drawing action, which normally corresponded to a line. No errors where made by participants while drawing.

All the 11 participants in the drawing-program experiment claimed that they were familiar with undo, but only 4 had used a version control system.

All the participants had used a drawing program, with Microsoft Paint being the most commonly drawing program, with 9 participants saying that it was their usual editor.

Subjective Measures Participants thought that the tree error-recovery method was easier to use compared to the other three error-recovery methods. After using each error-recovery method, the participants responses to the statement "It was easy to correct errors using this method" were measured on a five-point Likert scale, with 5 being agree. The mean rating for the tree was 4.2 ($\sigma = 1.0$), while the next highest error-recovery method was undo visualisation at 4.1 ($\sigma = 0.5$) followed by undo (3.9, $\sigma = 0.9$) and forward error-correction at 2.1 ($\sigma = 1.2$). Only the forward error-correction rated below the mean of 3.6 ($\sigma = 1.3$).

In addition, 7 of the 11 participants ranked the tree as their preferred interface, and all the participants claimed that they would use the tree if given the opportunity. All but two of the participants ranked the forward error-correction interface last, and the exceptions ranked it third.

5.2.8 Experimental Concerns

There are five concerns that exist with the experiment: the editing tasks may not be realistic, the tasks may be biased towards a particular error-correction method, the design of the interface may influence how participants solve tasks, the background of the participants may bias the experiment, and the results may not be applicable outside the drawing and text-editing domains. I discuss these concerns in this section, along with the measures that were taken to address these problems.

The most major concern is that the tasks that were used in the experiment are not realistic. For example, the participants rarely copy-type English prose: frequent editing tasks that were cited by the participants were composing email messages and creating computer programs. Likewise, the participants do not regularly copy drawings by freehand. For the typing and drawing tasks this should affect all error-correction methods equally. Likewise, the linear error-recovery task should be uneffected by the user having to copy text. However, using the forward error-correction, undo, and undo

visualisation error-recovery methods to perform branched error-recovery requires the participant to recreate part of the document by either typing or drawing. As the participant has carried out this part of the task before (in the second creation task) then he or she should be familiar with the text that has to be typed or image that has to be drawn, so the effect may be mitigated somewhat.

The tasks may be biased towards one type of error-recovery method. As mentioned in Section 5.1.5, forward error-correction is theoretically the fastest method of correcting single-character errors. However, it becomes progressively slower than the other error-recovery methods when longer passages of text need to be corrected — unlike the tree-based error-recovery method, which requires near constant time (Figure 5.3). However, for small error-recovery tasks, forward error-correction is clearly superior to the other error-recovery methods (Section 5.1.5) so there is little reason to test short passages of text. Larger errors should be harder for users to correct, which is why this experiment concentrated on these. However, linear and branched error-recovery occurs in equal proportions in the experiment, which may affect the participants subjective preferences in favour of the error-recovery methods that allow for a less common-task to be completed faster.

Using the mouse to start the timer in the text editor interface (Section 5.2.3.1) may influence how the participant deletes text when he or she is required to use forward error-correction. As the participant's hand is already on the mouse, he or she may be more likely to select the text using the mouse before and deleting it, rather than using the keyboard to select the text. It is estimated that using the mouse would make linear error-recovery and branched error-recovery tasks 1.2–1.3s faster than using the keyboard, but this may not be an accurate indication of how long a participant would normally take to carry out a similar text-editing task outside the experimental environment. However, the open text-editing task would give an indication of which method that participants preferred using.

The time taken by the participants may be faster than for the general population as the participants were predominately computer scientists. A small sample of participants from outside the computer-science field were used to indicate if there was a bias caused by the sample, and none was

detected.

Finally, the results may not be applicable to applications outside the drawing and text-editing domains. However, the domains are quite different, so this concern should be mitigated.

5.2.9 Discussion

The measured typing speed (Section 5.2.7.1) was similar to the estimated speed used in the keystroke-level analysis in Section 5.1, so in this section the estimated times in Table 5.7 are compared directly with the measured times. In addition, the mean typing speed for the open typing task (3.8 characters per second) was close to the mean typing speed in the initial copy-typing tasks (4.0 characters per second) so it appears that the cognitive effort involved in creating a document, or lack thereof, had little effect on the typing speed of the participants.

Forward Error-Correction The participants generally did not complete the typing subtasks without error, as is indicated by Table 5.8; forward error-correction was used to correct all these errors. The use of forward error-correction to correct small errors indicates that theses errors were easier to correct with forward error-correction than undo or either of the visualisations, as predicted.

Participants were significantly more likely to use the mouse to complete the error-recovery tasks using forward error-correction in the typing experiment; as mentioned in Section 5.2.8, this may be due to the participant starting the task with his or her hand on the mouse. However, the time taken by participants to carry out linear error-correction with the mouse (3.5s) was even faster than the predicted time (5.3s). This indicates that the participants were experienced at carrying out linear error-correction in a text-editor using forward error-correction with a mouse. The reason that the measured time was lower than the predicted time maybe attributed to participant not needing to home his or her hand on the mouse (saving 0.4s) and the fact that not all the mental-preparation time was measured (saving up to another 1.4s).

When used in the drawing experiment, the forward error-correction interface was slower than the other three error-recovery methods. The participants also ranked it poorly, with no participant ranking the interface above 3. This contrasts with the rankings in the typing experiment, where forward error-correction was ranked first by 5 participants, the second highest number of top-rankings after the tree interface, which received 11.

The results indicate that the effectiveness of forward error-correction was tied to the type of document that was being edited. A typing error was easy to recover from using forward error-correction. In contrast, a drawing error was comparatively slow and difficult to remove using forward error-correction.

Undo The results from the evaluation of the undo interface should be applicable to systems other than the text editor and drawing program, as the undo system was very similar to that found in common editing programs such as Microsoft Word. In addition most participants claimed they were familiar with the type of undo that is found in such systems. As all the participants had mental preparation time before beginning the error-recovery tasks, the possible cognitive difficulty of using undo should have been lessened.

In the text-editor experiment participants were slower at completing the linear error-recovery task using undo than with forward error-correction, despite the measured mean speed (5.6s) being slightly slower than the theoretical expert performance using undo (5.1s). The mean time to complete branched error-correction in the typing experiment (24.9s) was also very close to the estimated time of 25.4s. However, undo did not rate very highly in the subjective measures, with only 1 of the 21 participants in the typing experiment ranking it first.

In the drawing experiment undo fared better. It was the fastest method of recovering from linear-errors, and it was the second-fastest method for recovering from branched errors. In the subjective measures 2 of the 11 participants ranked undo first, and it had a mean rating above average (though it was second to last).

While the difference in the time taken to complete the error-recovery tasks was expected, the difference between subjective measures in the two experiments was not. The difference may be due to two reasons. First,

if completing a recovery task using forward error-correction is complex, as it was in the drawing experiment, then undo will be more beneficial than if forward error-correction is simple. Second, if the number of undo nodes created is high, as it was in the typing experiment, then `Control-z` will need to be typed many times, which may be less convenient than using forward error-correction.

Undo Visualisation The time that participants took to complete tasks using the undo visualisation error-recovery method was very similar to the speed of using undo in both experiments. The mean time for linear error-recovery in the typing experiment (5.4s) was slightly slower than the theoretical speed (4.2s), but the speed difference was similar to that between the estimated and measured performance of participants using undo. Participants took less time to complete linear error-recovery in the drawing experiment (4.7s) than the typing experiment (6.0s), even though the times should be comparable. This may be due to participants creating fewer undo-nodes in the drawing experiment.

The time taken to complete branched error-recovery using undo visualisation in the typing experiment was close to the estimated time, taking 25.5s rather than 23.6s. The difference in time between the branched and linear error-recovery tasks was due to the time taken to retype the text in the branched error-recovery task. There was a similar increase in the time taken to complete the branched error-recovery task in the drawing experiment (14.6s), which was due to the time taken to redraw parts of the image.

In the subjective measures, participants tended to rate the undo and undo visualisation interfaces similarly in both the typing and drawing experiments. This may be due to the interfaces being very similar, as both create states in the same way and neither allow branched error-correction. While the visualisation may have appealed to some participants because it was new, it was also hampered by the participants being unfamiliar with the interface.

Tree The tree error-recovery method was liked by the participants, and generally allowed them to complete the linear and branched error-recovery tasks quickly. For the linear error-recovery task in the typing experiment, the

tree was the second slowest interface, taking participants 5.9s to complete the task. This was slower than both the theoretical speed (4.2s, calculated using the formula in Table 5.6) and the speed that participants took to complete the same task using the undo visualisation method(4.4s). This difference may seem large, but the estimated time was still within a standard-deviation of the mean ($\sigma = 2.1$).

In the drawing experiment the tree allowed participants to complete linear error-recovery in the second fastest time, taking 4.4s compared to 3.7s for undo, which was the fastest error-recovery method. One possible reason that the drawing experiment produced a different result to the typing experiment was that there were fewer states in the drawing experiment, so there was less horizontal distance to move the marker.

For branched error-recovery, the tree was consistently faster than the other error-recovery methods in both the typing and drawing experiments. This was due to the fact that the other three error-recovery methods required the participants to recreate part of a document, while the tree did not. As with linear error-recovery, the time taken to complete branched error-recovery should be the same across experiments, but in the typing experiment the time taken by participants was 6.5s, compared to 4.4s in the drawing experiment. Interestingly, 4.4s was also the time taken to complete the linear error-recovery task in the drawing experiment. This indicates that participants found that branched error-recovery in the drawing experiment to be easier. This may be due to the fact that fewer states were created by participants in the drawing experiment compared to the typing experiment, or an anomaly caused by having a small number of participants in the drawing experiment.

In both experiments the standard deviation for branched error-recovery was larger than for linear error-recovery. This may be due to some participants becoming confused with the branching and taking longer than expected to complete the task; other participants completed the task in less time because the state was easier to find, as it was at the end of a branch.

5.3 Conclusion

In this chapter I presented a theoretical and empirical evaluation of systems that support error-recovery. Despite being unfamiliar with the tree error-recovery method, participants completed the complex branch error-recovery task ("change of heart") in the shortest time in both the typing and drawing experiments. In addition, participants came close to completing the tasks in the theoretically optimum time. The tree error-recovery method was also generally preferred over the other three error-recovery methods in the subjective measures.

For the simpler (linear) error-recovery task, forward error-correction was the fastest method in the typing-editor experiment, while undo allowed the participants to finish the equivalent task in the shortest time in the drawing-program experiment.

I recommend that systems provide a combination of error-recovery methods. A visualisation can be provided that would allow the user to complete the branched error-recovery task, while a combination of undo and forward error-recovery can be provided to allow the user to complete linear error-recovery. I suggest that stack-based undo be implemented as it is simpler and more common than history undo. (In practise, undo would move back along the undo tree, while redo would select the most recently created branch.)

I suspect the *mechanical* disadvantages of undo are sufficient to reduce its use in a text editor to a level below that of forward error-correction. This is not to say that undo does not cause a breakdown because it is cognitively complex, but the mechanical disadvantage to using undo is sufficient to account for the relatively low use.

I undertook the study presented in this chapter in order to gain an insight into the second of my research questions: what is the best way to use time to organise different versions of the same document? Overall, the tree turned out the be the most effective method in the evaluation. However, the evaluation only provided an initial answer to this question for four reasons. First, not all error-recovery interfaces, or editing environments, were assessed — so some error-recovery mechanisms may work particularly well when editing particular types of document. Second, I did not assess the cognitive difficulty

of using each interface, which may affect how quickly and easily a person is able to recover from an error. Third, the retrieval tasks the evaluation presented here involved the recovery from errors that took place in the recent past, and the interfaces may preform quite differently if the user needed to retrieve a document from the distant past. Finally, the performance of the error-recovery mechanisms may be different in a computer-supported collaborative workspace, where multiple users are editing the same document.

There are other questions that stem from this evaluation.

- Can the usability of the tree be enhanced by using more a complex visualisation? For example, visualisation for the GIMP provides the user with a thumbnail of the image at each state (a description cue). Would such a system create too much clutter, reducing usability, or allow the user to find states more quickly? What other features could be added to the tree?

- Would a tree-based visualisation be used outside the lab? The usability of the tree is good in theory, and the empirical evaluation of the tree also showed that it is usable. However, the tree may not be *useful.*

In Chapter 6 I conduct an evaluation of a text editor that uses the tree as the main method for providing access to documents and document versions. The evaluation aims to answer some of the above questions.

Chapter VI

Swaca

In this chapter I present the design and evaluation of a document organisation system that *only* uses time to organise documents, without the need for document names and folders. As well as organising documents, the system, Swaca[1], organises versions of documents using the same interface. I show that the system is both useful and usable, with users able to find documents and correct errors without confusion. The study found that users preferred their existing document-organisation systems to Swaca, but this was expected as those systems are more refined and the participants have more experience with them.

In this chapter, I consider the retrieval of documents that are organised by time and the retrieval of document versions together — unlike the prior chapters where the two research themes were mostly treated separately. My primary motivation for creating and evaluating Swaca was to see if a temporal document-organisation system that did *not* use files was useful and usable. A formative study was used to assess Swaca, with the aim of discovering the usability problems with the system, and how it was used outside the overdetermined environment of an experiment.

While there is no control in the empirical evaluation presented in this chapter, the small-scale study did allow me to form some guidelines for the design of temporal document-retrieval systems. I begin with the design of Swaca in Section 6.1. The evaluation is then presented (Section 6.2). Finally, this chapter is concluded in Section 6.3.

[1] System With A Cool Acronym

6.1 Design of the Swaca Text Editor

The guidelines presented in the prior chapters informed the design of the Swaca text editor (Sections 2.1.4, 2.2.3, 2.3.3, 2.4.1, 3.1.3, 3.2.6, 3.3.3, 4.5, and 5.3). I used an iterative design to create Swaca: first a paper prototype was created and tested (Figure 6.1(a)) before I coded the complete system (Figure 6.1(b)).

There are two main components to the Swaca interface: the text-entry area, which is used for editing (Section 6.1.1), and the timeline that is used for some error-recovery tasks and document retrieval (Section 6.1.2). In this section I discuss both of these, before the implementation of Swaca is discussed in Section 6.1.3.

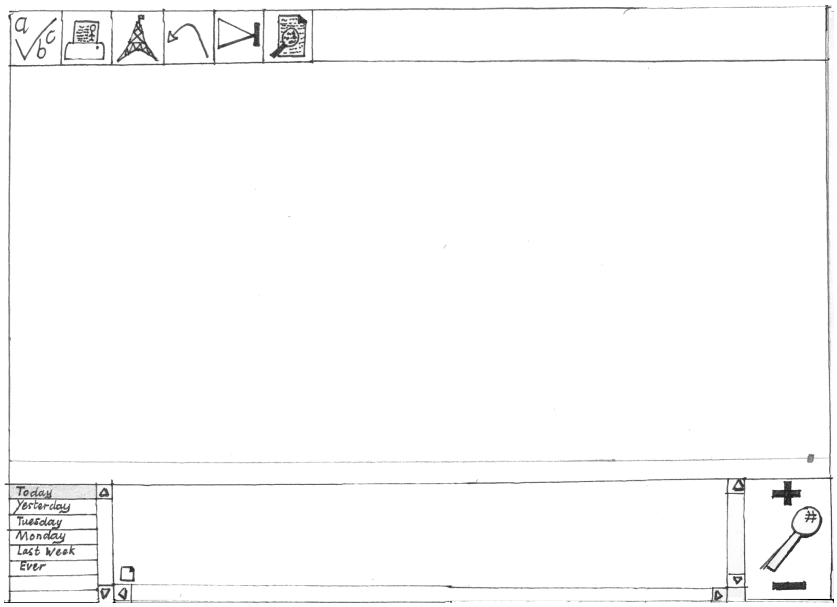
6.1.1 Text-Entry Area

The text-entry area in Swaca is broadly similar to text-entry areas in other editors: the keyboard-navigation keys move the text-entry point, pressing `Control-z` invokes undo, `Control-y` redo, holding the `Shift` key while moving the text-entry point selects text, and `Backspace` deletes the selected text (or the character to the left of the text-entry point if no text is selected).

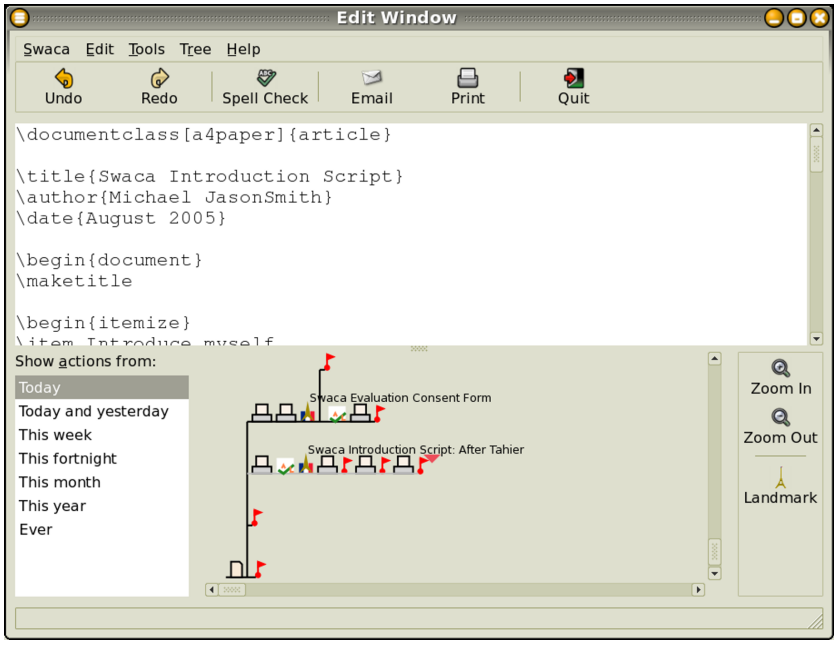
I used the interface guidelines for the Gnome desktop (Benson et al., 2004), along with some guidelines taken from Apple (2001) to inform the design of the text-entry area. The design guidelines did not state when to create an undo node when typing. In the text-editor that was evaluated in the undo experiment, an undo node was created when the user paused for 200ms. While using Swaca I felt that this period was too short, as it caused the tree to become too big, and the amount of text that was changed when undo was invoked was too small, so the period was lengthened to 400ms (Figure 6.2).

6.1.2 Timeline

The timeline is based on the tree-based error-recovery method, which was evaluated in Chapter 5. As the user carries out actions they are marked on the timeline. A marker indicates the current state; this marker can be moved



(a) Paper prototype of Swaca.



(b) The finished Swaca prototype. The timeline has been zoomed out one level, so the typing and deleting actions are not shown, and only the actions carried out in the last 24 hours are displayed. The red marker points to a position after the pause-flag on the third branch from the top, which also has the landmark named "Swaca Introduction Script: After Tahier".

Figure 6.1: The paper-prototype and finished version of the main Swaca interface. The window is split in two: a text-entry area at the top and the timeline at the bottom.

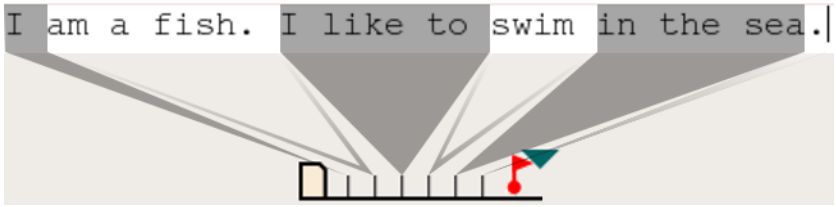


Figure 6.2: An example of the relationship between typed text and the undo nodes that are created, represented by tic marks on the timeline. The eleven-word phrase is the same one that participants wrote in the earlier undo-experiment (Section 5.2), while each of the six undo nodes were created after the user paused for at least 400ms.

to retrieve a prior state. When the user retrieves a prior state and carries out an editing action a branch in the timeline is created.

There are three major differences between the timeline in SWACA and that evaluated in Chapter 5. First, icons are used to mark 'important' actions in SWACA: spell-checking, emailing, printing, pausing and quitting the system. Second, the user is able to explicitly mark specific points on the timeline with 'landmarks'. Important actions and landmarks were added to assist with reminding and document retrieval.

In this section I will look at how the timeline in SWACA supports reminding (Section 6.1.2.1), document retrieval (Section 6.1.2.2) and error-recovery (Section 6.1.2.3). In Section 6.1.2.4, I examine the other features that were present in SWACA.

6.1.2.1 Reminding

An important task for document organisation systems is reminding the user of his or her current tasks (Section 2.3.3). In SWACA, the recently completed actions, which are displayed on the timeline, act as a reminder of the user's current tasks. The actions should act in a similar way to piles of documents in the physical world.

To further support the reminding tasks, the actions that were performed in the last 48 hours were displayed when SWACA is started. The timeline is also zoomed out to begin with, so the typing and deleting nodes are not shown, and the marker, which indicates the current point that is displayed

Title	Creator	Subject	Description	Date	Format	ID
✔		✔	✔	✔		

Table 6.1: The retrieval cues that the timeline in SWACA supports. A ✔ indicates that a cue is supported.

in the text editor, is positioned next to the most-recent action (which is a node that indicted that the user quit SWACA).

In early versions of the system, actions from the last 24 hours were shown when SWACA started. However, this was changed as I found that too few documents were shown.

6.1.2.2 Document Retrieval

SWACA primarily offers a temporal cue to document retrieval: the order that events occurred. People are better at recalling the order of events than the date of an event (Section 2.2.3) so dates are not shown on the timeline, but moving the mouse over a node retrieves the date that the node was created (Table 6.1). Icons, representing the user's actions, are shown on the timeline to provide a description cue. Typing and deleting actions are shown as tic marks, icons that are similar to those used in the toolbar represent printing and spell-check actions.

To indicate the end of typing episodes (Section 3.2.5.2), small red flags are placed on the timeline when user pauses or quits SWACA. (In early trials of the system it was found that the red timeline position-marker was had to see among the red pause-nodes, so the marker colour was changed to green.) The period of time before the pause-marker was placed on the timeline was altered during the iterative design of the system: too short and the timeline became covered in small red markers, too long too few flags appeared. In the end a delay of two minutes was found to be appropriate.

To assist with recall, information about any node can be accessed by clicking on the node, which opens a window that lists information about the

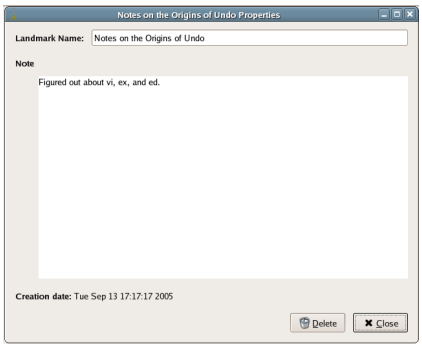


Figure 6.3: A properties window for a landmark in SWACA — specifically the landmark second from the left on the bottom branch of the timeline shown in Figure 6.4(a). The landmark has a name ('Notes on the Origin of Undo'), a note ('Figured out about vi, ex, and ed'), and creation date associated with it.

node (Figure 6.3).

Landmarks are named points on the timeline that represent named documents or named document-versions. Unlike other nodes on the timeline, which are created implicitly as the user edits the document, landmarks are explicitly created by the user: clicking the landmark icon to the right of the timeline, which looks like the Eiffel Tower, causes the system to prompt the user for a name and a textual description the landmark. The name is displayed on the timeline next to the icon of the landmark; the description is retrieved when the user clicks on a landmark icon. Landmarks should provide the same advantages as the explicit version-creation systems, such as RCS (Section 3.3.2.1), because the user explicitly creates landmarks, so there should be fewer landmarks than the other icons on the timeline.

In early version of SWACA, there was no default name for the landmark. However, I found that I created many landmarks with the same name, as they indicated different *versions* of the same document. Therefore I changed the system so the name of a landmark defaults to the name of landmark immediately prior to the current one, or the subject line of an email message if that is more recent. This meant that, to create the landmark, the user did not have to preform any actions other than accepting the default name.

'Semantic zoom' (Perlin and Fox, 1993) was implemented because I an-

ticipated that there would be a large number of nodes displayed on the timeline. When the user zooms out, less important nodes are removed from the timeline, which should aid retrieval of more important document states. To further reduce the amount displayed, branches that do not have any nodes on them are also removed. However, this policy can lead to the current branch being removed from the timeline, which is an undesirable situation. Therefore, the current branch is always displayed, no matter the zoom level.

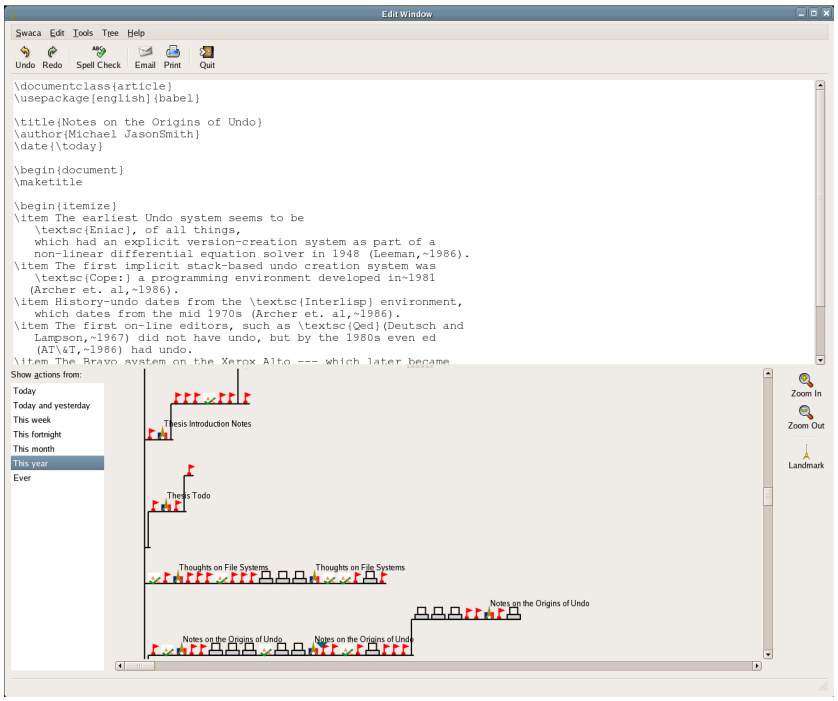
Finding Old Documents Following the findings of Barreau and Nardi (1995) and the recommendations in Section 2.3.3, it is particularly easy to find recent documents in Swaca. However, it is possible to find older documents.

Finding an older document generally starts with changing the temporal chunk, so the older documents are displayed on the timeline (Figure 6.4(a)). A simple list, located to the left of the timeline, is used to provide temporal chunks, as participants in the history-list experiment preferred this (Section 4.5). The temporal groupings are based on linguistic norms, following an idea from Larsen et al. (1999): today, today and yesterday, this week, this fortnight (14 days), this month, this year, and ever. The groupings have the added advantage of providing group-names that do not change, and the names always suggest time-periods in the past (unlike 'Monday' for example).

In early trials of Swaca, I found that the first action that was carried out after changing the temporal chunk was zooming out, so more of the timeline could be seen. Therefore I altered the behaviour of the system, so the timeline was zoomed out whenever the chunk was changed. The timeline was automatically zoomed in, so all the nodes were shown, when the document was edited. Because of this, I found that I rarely needed to explicitly use the zoom feature.

After early trials of the system, the ability to double-click icons to move the marker to that point was added, as the timeline can become too large to drag the marker around easily. Moving the marker this way was quite similar to opening a document in a desktop file-manager.

In my experience using Swaca, I found that most documents were not based on earlier ones, with most of the branching caused by error-recovery



(a) Swaca with the timeline zoomed out one level, so the typing nodes are not shown, and the 'This year' chunk selected.

(b) A portion of the timeline shown above, zoomed in fully. The branch has been caused by an undo, which recovered from two typing actions. The lower branch only contains typing and deleting nodes; the upper branch displays three printing nodes, two pause nodes, a landmark, another pause and a final printing node, in addition to the typing and deleting nodes.

Figure 6.4: A view of four documents in Swaca: Thesis Introduction Notes, Thesis Todo, Thoughts of File Systems, and Notes on the Origins of Undo. The branches present on three of the four documents are caused by undo actions, as shown in the segment of timeline in Figure 6.4(b).

tasks (Figure 6.4(b)). However, the structure that is created by the edits does provide the user with some spacial cues.

6.1.2.3 Error Recovery

In Swaca I have designed a system that combines implicit and explicit version-creation, and can be used to complete linear and branched error recovery. It is hoped that the usability and utility of the system will be greater than those that keep implicit and explicit version-creation separate.

Recovering from errors using Swaca is done in three ways:

1. Forward error-correction,

2. Using undo and redo, or

3. Manipulating the marker on the timeline.

Forward error-correction is carried out using the text-entry area; following the results of the evaluation of error-recovery methods (Section 5.2.7.1), it is anticipated that it will be used to correct errors in typing when the number of actions to undo is fewer than six.

Undo and redo are accessed using either the keyboard or the toolbar. They are mapped to the timeline: undo moves the timeline-marker back along the tree (towards the root) while redo moves the marker towards the leaves, selecting the most recently created branch when there is any ambiguity.

The timeline marker can be dragged (using direct manipulation) to undo multiple actions, as recommended in Section 3.3.3. This should allow the user to complete branch error-recovery more quickly than is possible with undo alone, as shown in the evaluation of error-recovery methods (Section 5.2.7.1).

In my experience using Swaca, documents are generally organised vertically, while versions are organised horizontally (Figure 6.4(a)). Finding a particular version of a document is a two-step process: scrolling vertically to find the document (normally using the names of landmarks) and then scrolling horizontally to find the version of the document that is of interest. A single document will generally have multiple landmarks, which indicate significant versions; marking versions with landmarks also ensures the branch

has a name on it after the prior landmark has been moved out of the default chunk ('Today and yesterday'). The landmarks can have short descriptions of the version, indicating what is important about the state of the document, as shown in Figure 6.3.

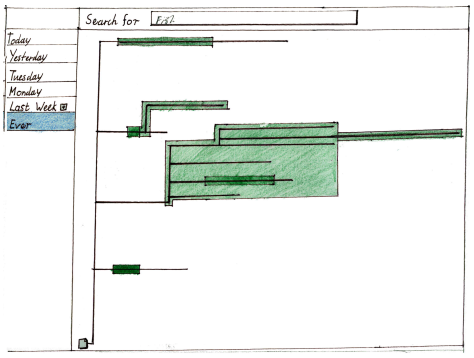
6.1.2.4 Other Features

Two features of the timeline were trailed and abandoned in early designs of Swaca. The first was to continually grow the timeline, so the gaps between typing and deleting nodes represented the pauses that the user made while typing. (If the user paused for a long period of time the line stopped growing.) While this did clearly show the user's typing episodes, it quickly caused the timeline to become large, and made dragging the position marker hard, as the distance that the marker needed to move to undo an action was larger. The second feature of the timeline that was trialled was changing the size of the typing and deleting nodes in proportion to the amount of text that had been changed. The benefit was that it made the typing and deleting nodes easier to click on, and it provided an extra retrieval cue. However, like the prior feature, this caused the timeline to become large in a short space of time, and made it harder to use the timeline for error-recovery, as the marker needed to be dragged further. In addition, I felt that the node size did not provide an effective cue to document retrieval.

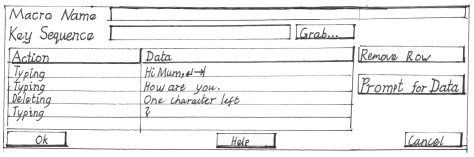
Two features were developed in the paper prototype but were not implemented in the final system: searching and macros (Figure 6.5).

As there are no named files, Swaca cannot return a list of filenames that match a search-string, unlike most other full-text retrieval systems (Blair and Maron, 1985; Cunningham and Connaway, 1996; Freeman, 1997; Brin and Page, 1998; Witten et al., 1999; Pitkow et al., 2002; Yahoo!, 2005). However, I have devised a scheme in which the timeline can be used to display *areas* of branches that match search-terms (Figure 6.5(a)). The density of colour indicates how many search terms matched, and the temporal groups to the left of the timeline is used to further refine the search.

To create a macro, the user selects a sequence of actions from the timeline; this forms the basis of the macro that is then edited in an interface similar to



(a) The prototype search interface showing the branches that match the search term 'fish'. The density of colour represents the strength of the match is indicated.



(b) The prototype macro-editor, showing a sequence of actions that had been selected from the timeline.

Figure 6.5: Two features that were not present in the finished program: searching and macro creation.

that outlined in the paper prototype (Figure 6.5(b)). For simplicity, neither macro-creation or searching was present in the finished Swaca system.

A third feature was discussed, but has not been developed in any prototype: deleting nodes. Unlike a standard system, there is no way to destroy text once it has been written into Swaca. A deletion system would involve moving nodes into the equivalent of the 'trash can', where they would stay until the user explicitly removed them from the system. Rather than overly complicate the system, it was decided not to implement the deletion of nodes, but it was expected that participants would want to delete nodes.

6.1.3 Implementation of Swaca

Swaca is written in Python (van Rossum and Drake, 2005), using the Gtk+ gui toolkit (Clasen, 2005). The main advantage of Gtk+ is its use as the default toolkit for the Gnome desktop, so Swaca visually matches the majority of the user's applications. In addition, high-level widgets, such as a text-editor and canvas, are used to provide much of the functionality in Swaca.

While Swaca does not provide a view of a folder hierarchy, it is necessary to store data using the standard file-system. This is done by writing to a

hidden file in the user's home folder. The file-format itself is a serialised form of the internal Python data-structure that represents the user's work. Every 31 seconds, if the user modifies the state of SWACA, the code spawns a new thread that serialises the appropriate internal data-structures to the file-system. A new thread is spawned so the user is not interrupted when the data was stored.

6.2 Evaluation of Swaca

In this section I present a small-scale study of SWACA, similar to the evaluations described by Nielsen and Landauer (1993) and Nielsen (2000b). The evaluation centred around a structured interview of participants that used SWACA for a fortnight. The interview sought to assess error-recovery and document organisation in SWACA: discovering the benefits and pitfalls of the system when used outside the lab.

In this evaluation, I do not directly compare one interface against another — unlike the earlier studies presented in this thesis (Chapters 4 and 5). This is because existing document-organisation systems, such as file systems, have been in development and use for the last forty years (Daley and Newman, 1965), while SWACA has had little refinement, and the participants have had little experience with the system.

The participant details are given in Section 6.2.1, before the apparatus and procedure used in the evaluation is presented (Sections 6.2.2 and 6.2.3). Concerns with the evaluation are then discussed (Section 6.2.5). Section 6.2.4 details the results, before the discussion in Section 6.2.6.

6.2.1 Participant Details

Seven volunteer postgraduate students in the Department of Computer Science participated in the study, for which there was no reward.

6.2.2 Apparatus

All machines in the study used Linux as the operating system and ran a Pentium 4 CPU. However, the speed and graphics capabilities of the machines

used in the study varied between participants as each used his or her own machine. The SWACA data store, as with most of the participants' files, was kept on a central file server that was accessed using NFS over a switched 1Gb full-duplex Ethernet network.

6.2.3 Procedure

SWACA was introduced to each participant in a fifteen minute one-on-one tutorial. First, the reason for conducting the evaluation was explained. The main features of the text-editor component of SWACA were demonstrated by asking the participant to type a short passage of text and then checking the spelling. Emailing and printing was then demonstrated. The features of the SWACA timeline were then introduced: the participant was shown how to navigate the timeline, zoom in, zoom out, and change the temporal chunks. In addition to timeline navigation, the relationship between undo and the SWACA timeline was discussed before the participant was shown how to access the on-line help (Appendix A). Finally, the participant was shown the data that was gathered as the system was used.

Each participant was given a fortnight to use SWACA. There was no compulsion to use SWACA, the participants could opt out of the evaluation at any time and were free to use any other program during the study.

At the end of the evaluation, a structured interview was conducted with each participant (Appendix B). Statistical information was also gathered from the data-store maintained by SWACA, and a log of the participant's use of undo, redo and timeline-manipulation was retrieved.

6.2.4 Results

Of the seven postgraduate students who volunteered for the study, six completed the evaluation of SWACA. The data from that participant that did not complete the evaluation has not been included in the results. The other participants were interviewed sixteen to twenty days after being introduced to the system.

The gathered results fell into four areas:

1. Background information about the participants (Section 6.2.4.1),

Editor Type	n	**Rating**	σ
Text Editor	6	3.6	1.4
IDE	3	3.3	1.2
Web Browsers	2	3.0	1.4
Presentation Program	1	3.0	—
Word Processors	3	2.5	1.2
Email Clients	2	2.5	0.7
Image Editor	2	1.5	0.7
Diagram Editors	2	1.5	0.7

Table 6.2: "Normal" editors used by the participants in the evaluation, according to category. The second column lists the number of *distinct* editors; the rating is for all editors in the category, as measured on the scale never (1) to always (5).

2. The participants' opinions on error-recovery (Sections 6.2.4.2),

3. Opinions on document organisation (Section 6.2.4.3), and

4. The participants' suggested improvements for SWACA are listed in Section 6.2.4.4.

6.2.4.1 Participant Background

The participants used a diverse number of document editors. When asked "What are your normal document editors?" the participants listed 21 different programs, a mean of 2.9 editors per participant ($\sigma = 1.3$). Table 6.2 lists the different editors, grouped by type.

When asked what sort of documents they create, the participants cited program source-code, papers, presentations, reports, Web pages, essays, free-form documents (for recording notes), and email messages. The participants' mean rating to the statement "I normally start documents from scratch, rather than basing them on an older document" was 2.8 ($\sigma = 1.2$), as measured on a five-point Likert scale (5 is agree). The participants frequently stated that LaTeX documents were based on earlier documents, by either copying an existing file or copying part of the file content. According to one participant, a reason for not employing 'templates' more often was the difficulty in finding documents, while another participant stated that, as profi-

ciency in editors increased, template files were used less. Another participant stated that it was simply easier to start documents from scratch.

The participants' favoured method of opening files was the command-line — where the name of the editor was written with the filename following — giving it a mean rating of 4.2 ($\sigma = 0.8$) on a five-point semantic differential scale (never–always). The use of "the file browser and the desktop" was the lowest rated method of opening files, with a rating of 2.3 ($\sigma = 0.8$). The use of the recent files list was rated at 2.5 ($\sigma = 0.6$), while the use of the open dialog was rated at 3.3 ($\sigma = 1.0$). When asked about their preference for the command line, typical statements from the participants were that a GUI file manager was not needed under Unix, the command-line was easier to use because it required fewer mouse-clicks, that many shells were open and readily accessible, and that a GUI file-manager was "just not me." However, three participants stated that, under Microsoft Windows, the file manager (Explorer) was used, while the command line (Command) was not. The same participants stated that the opposite situation was the case under Linux (with the GNOME Nautilus file manager and GNU BASH shell). Two reasons given for this disparity were that it seemed that Windows was built around the GUI (and conversely, Linux was not), and the command-line in Windows was not as good as BASH: "I hate the command `dir`."

The participants preference for keyboard-based interaction was also evident in their claimed use of undo: all stated that they used the keyboard to activate undo in their 'normal editor'.

6.2.4.2 Error-Recovery in SWACA

Participants favoured the keyboard when carrying out error-recovery using SWACA: they stated that "normal undo" (using the keyboard) was used more than the tree, with the mean rating for the former of 3.0 ($\sigma = 1.3$) and 2.0 ($\sigma = 1.6$) for the latter, as measured a five-point semantic differential scale (never–always). However, the participants attitude toward undo was neutral: when asked if they agreed with the statement "The tree was better than undo" the participants gave a mean response of 3.5 ($\sigma = 1.1$), as measured on a five-point Likert scale. When using the tree, the participants typically

dragged the marker, rather than double-clicking on nodes, with 5 stating that they dragged the marker, and one stating that the tree was never used.

Three participants said that they tended to use forward error-correction (Section 5.1.1) rather than undo to correct errors. Two stated that they used undo to remove a sentence or part of a sentence, while one could not remember what undo was used for (but it was used). Another stated that undo was not used at all because editing was done outside SWACA and pasted into the system — effectively using SWACA as a standard version control system (Section 3.3.2.1). The tree was used to get rid of entire paragraphs, or "a stack or characters," according to three participants, while another used it more for exploring the system.

6.2.4.3 Document Organisation

Overall, the participants did not find document retrieval in SWACA particularly easy or hard: the mean Likert-scale response to the question "It was easy to find documents using SWACA" was 3.2 ($\sigma = 1.5$). The mean response to the statement "the lack of files was confusing" was 2.3 ($\sigma = 1.4$). As expected, they did not think that SWACA "was better than my normal method at finding documents", with a mean response of 2.2 ($\sigma = 1.0$) on a five-point Likert scale. A telling comment from one participant was "[I am] so used to files."

Three participants stated that they tried to save while using SWACA, out of "reflex" or "ignorance", while another stated that there was no desire to invoke save. Participants also stated that not saving was "nice" and "not confusing, once you were used to it." Overall, the mean response to the Likert-scale statement "I wanted to invoke save while using SWACA" was 2.7 ($\sigma = 0.8$); the participants mean response to the Likert-scale statement "I was confident that data would not be lost" was 3.7 ($\sigma = 1.4$).

When asked about the documents that were created in SWACA, the participants frequently cited note-keeping and 'to do' lists. Four papers were written in SWACA, and one participant used the system for maintaining a highly-structured file, similar to a flat-file database. When the participants were asked whether SWACA was still being used, the mean response was 2.1

($\sigma = 0.8$) as measured on the five-point semantic differential scale never–always (5). Reasons for not using SWACA included the lack of file-system support, no need to create new documents during the trial period, not much use for a text-editor (relying on an IDE and word processor instead) and commitment "to my other tools."

When using SWACA, the participants started documents from scratch more often than with their "normal" document editors: the mean rating in response to the statement "I normally start documents from scratch, rather than basing them on an older document" was 3.5 ($\sigma = 1.64$) for SWACA, compared to 2.8 ($\sigma = 1.2$). The participants also claimed that the documents they had created in SWACA were smaller than normal, with a mean rating of 2.7 ($\sigma = 0.8$) as measured on a semantic differential scale of 'smaller' to 'larger' (5).

Branches on the timeline can be created as part of error-recovery, or when a new document is created. While the dual-role of branches did not confuse, the difference between the two types of branches was not particularly clear to the participants: 3.5 ($\sigma = 1.4$) was the mean response to the statement "I could readily distinguish between 'corrections' and 'documents'." The mean number of branches created by the participants was 8.8 ($\sigma = 4.1$), of which a mean of 4.8 branches ($\sigma = 1.8$) were branched off the root node (Table 6.3). Overall, participants performed a mean of 573.7 actions ($\sigma = 420.2$), which equates to 64.9 actions per branch.

The mean response to the statement "landmarks helped me find documents in SWACA" was 4.7 ($\sigma = 0.5$), indicating that landmarks were a useful feature. However, the participants did not think that "landmarks were better than named files", with a mean response of 3.0 ($\sigma = 1.1$). Three participants cited the creation of notes, associated with landmarks, as a useful feature, with one stating that note creation was used "a couple of times." One participant stated that landmarks were created instead of save, another simply stated that "making a landmark helped me", while a third said that "[I] don't see much difference" between landmarks and files.

Icons on the timeline were generally met with a neutral attitude by the participants. The mean response to the statement "The email, print and spell check icons on the timeline helped with finding documents" was 3.3

	S1	S2	S3	S4	S5	S6	Mean	σ
Branches								
Number	5	15	11	8	4	10	8.8	4.1
% off Root	*80.0%*	*46.7%*	*27.3%*	*87.5%*	*75.0%*	*50.0%*	*54.7%*	*23.3%*
Actions								
Typing	797	307	280	641	8	359	398.7	280.7
Deleting	380	110	103	259	1	72	154.2	139.1
Pauses	14	17	13	18	4	16	13.7	5.1
Emailing	2	0	4	2	0	0	1.3	1.6
Printing	0	0	4	0	0	0	0.7	1.6
Spelling	3	3	0	3	1	3	2.2	1.3
Landmarks	2	6	2	2	3	3	3.0	1.5
Total	*1,198*	*443*	*406*	*925*	*17*	*453*	*573.7*	*271.3*
Actions per Branch								
	239.6	*29.5*	*36.9*	*115.6*	*4.3*	*45.3*	*4.9*	

Table 6.3: Statistics gathered from the datafiles of the participants in the SWACA evaluation.

($\sigma = 1.2$). Participants stated that checking the spelling of the document was often carried out as a proxy for save, and to indicate that a subtask had been completed. Others stated that the structure created by the icons was useful, and it helped in remembering prior tasks. However, one participant stated that the icons were not used.

Zooming and temporal chunks came in for negative criticism: the mean ratings — of 2.8 ($\sigma = 1.2$) and 2.7 ($\sigma = 1.4$) respectively — indicate that the participants did not think that these two features "helped find documents on the timeline." Maintaining the length of the currently-viewed branch when zoomed out was not seen as a feature by some participants, as the branch was often too long and prevented the rest of the tree from being seen. Another thought it was "weird" how the current branch was always shown. One participant stated that zooming caused documents to become "lost" more than found. On the positive side, another thought that zooming helped because otherwise the tree was "too big."

The temporal chunks defaulted to showing actions carried out on the last two days, and this caused one participant to express difficulty in using the system, while another participant stated that the temporal chunks were never used. However, one participant like how the default chunk did not need to

be changed, while a fourth participant used the temporal chunks to find a document that was worked on "a week ago."

As can be seen in Table 6.3, participant S5 created fewer branches and carried out fewer actions than the other participants. S5 was the participant that, when questioned on how SWACA was used, stated that entering structured data into SWACA was not practical, because the system lacked a goto-line feature. Instead, S5 typed data into a separate editor and copied it to SWACA, which effectively acted as a version control system. This method of interacting with the system had the additional side effect of creating few icons.

6.2.4.4 Suggestions for Improvement

The participants suggested many improvements for SWACA.

- Fewer bugs, including a smoother implementation of dragging, and better tree drawing.
- More features for the text-entry area, including syntax highlighting, find and replace, and goto line.
- Scaling the scroll bars on the timeline to the size of the tree.
- Better overview and context, as zooming out removes the context, while zooming in removes the overview.
- Labels at the *start* of branches.
- File import and export was requested often.
- Storing the SWACA datafile in a non-binary format.
- User-defined metadata, such as classifications.

The participant who kept structured data in SWACA (S5) also suggested that metadata should be able to be associated with *parts* of the document, rather than particular versions.

6.2.5 Evaluation Concerns

The limitations with the evaluation were concerned with the lack of refinement, the inability to work with documents outside Swaca, and the lack of an experimental setup.

Swaca has had little refinement, unlike the editing and document-retrieval systems that it was compared against. This should have resulted in lower ratings for Swaca when compared to other systems.

The second concern is that the participants could not use Swaca to work with existing documents. I did not add the ability to import or export files, as the participants may have used this to organise documents, rather than the temporal mechanism. This should have limited the work the participants in the evaluation could carry out using Swaca.

There are three concerns with the evaluation that stem from the lack of an experimental setup. First, there is no control in the evaluation: participants were free to use Swaca as they chose. This means measured results from participants cannot be directly compared to each other because individuals could have been be using the system to perform completely different tasks. Second, the participants were a self-selected group from an academic department, whose background and knowledge may not match that of the general population. Finally, there are few participants in the evaluation, so one user could dramatically alter the reported means.

6.2.6 Discussion

Despite existing in an environment where many fully-featured text-editors are available, Swaca was used by the participants in the study, and they found that working without files to be useful and usable. Participants did not think that Swaca was better than their normal document retrieval systems, but this was to be expected as they have more experience at using their normal systems, and Swaca does not have the same level of refinement.

The participants in the evaluation had a bias towards command-line document retrieval interfaces. Specifically, the participants stated a preference was for the Gnu Bash shell, and a dislike of the Windows command-line. In addition, the gui-based document retrieval methods were given the lowest

ratings by the participants, while the command-line was given the highest. I suspect that the participants' heavy use of command-line interfaces, text-editors and IDE systems is not reflective of the wider community, and would bias the results against SWACA.

It appears that the participants adapted to error-recovery using the tree, and used it in an optimal way. When queried about the errors that were corrected using the different mechanisms, they stated that forward error-correction was used for small errors, while the tree was used to correct larger errors, of the size of paragraphs. The results from Chapter 5 showed that correcting small errors would be completed in the least time using forward error-correction, while larger errors are more quickly corrected with undo or tree-based error-recovery; the latter becomes the fastest error-recovery method when more than six words need to be corrected (Sections 5.1.5 and 5.2.9).

Landmarks were liked by the participants, with positive comments and high ratings. One possible reason for this is that they are a useful feature, allowing the participant to name a location on a branch. The note feature of landmarks — which allows a small description to be attached to the landmark — was also found to be useful, and this could have had a positive influence on the ratings. Another reason that landmarks found favour with the participants is that it provided a substitute for save. The neutral rating of landmarks when compared to files lends weight to the argument that landmarks were used as a proxy for saving, with one user explicitly stating that this was the case. A study of a system that provides save and landmark creation would provide further insight into the relationship between landmarks and save.

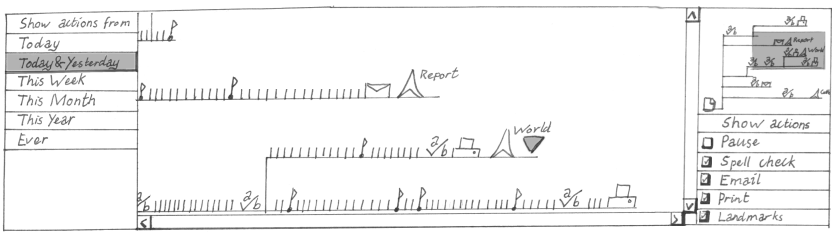
The icons, which indicate that a significant action had been undertaken, were given neutral ratings by the participants. However, comments indicated that the icons helped recalling prior tasks and indicated when a subtask had been completed, which were two goals of the system. The mean number of icons created was greater than the number of landmarks, which indicates that the zoom-levels were adjusted correctly.

There were two parts of SWACA that came in for consistent criticism: zooming the tree and the temporal chunking mechanism. The behaviour of the current branch and the lack of concurrent overview and detail were the

main criticisms of the zooming mechanism. I felt that it was undesirable for zooming to alter the state of the document, so when the user zoomed out the current branch was always drawn. However, the size of the branch often caused the remainder of the tree to be scrolled off to the left, so it could not be seen, which caused frustration for the participants, who had to scroll to see the other branches. Another criticism of the zooming mechanism was that the context of the branch could not be seen at the same time as the detail. A solution to both these problems would be to provide a 'gestalt view' of the tree. Gestalt views are found in many computer games, such as Sid Myer's Alpha Centauri (Figure 6.6(a)), Neverwinter Nights, and Fable (Firaxis, 2001; Infogrames, 2002; Microsoft Corporation, 2004). Besides games, gestalt views have been added to many other systems, including text-editors, electronic whiteboards, video annotators (Baecker et al., 1993; Edwards and Mynatt, 1997; Brown et al., 1998). More applicable to SWACA, the temporal document-organisation system TimeSpace used a gestalt view of the timeline, and it was liked by the participants in the observational study conducted by Krishnan and Jones (2005). A gestalt view of the tree in SWACA would provide the context of the branch that is currently being edited (Figure 6.6(b)). Documents or versions could be accessed using the gestalt view, while the standard tree view could be used for finer-grained editing. An alternative method of providing overview and context would be to distort the tree, using a fish-eye lens effect (Carpendale and Montagnese, 2001), but users should be more familiar with a simple gestalt view.

The temporal chunks were also criticised by the participants. I hypothesise that one problem with the temporal chunks is caused by infrequent use of SWACA. If the system is used infrequently, the documents that were last edited would not be visible, as they would have been moved into a chunk other than the default. This would be more noticeable after a weekend, for example: SWACA would not have been used for two days, so the documents that were last edited would not be visible when the participant started the system on Monday. Finally, the temporal chunks automatically managed the state of the tree, so the participants did not need to frequently interact with the chunks, as indicated by one participant; this would make the temporal chunks relatively unfamiliar compared to the ever-updating and highly visi-

(a) A screen-shot of Sid Myer's Alpha Centauri, a turn-based strategy game (Firaxis, 2001). The image shows the main view of the map, where the game is played, with a the details of units and locations; the gestalt view, in the bottom right of the image, provides context.



(b) A paper-prototype of Swaca with a gestalt view, shown in the top right of the image. The area highlighted in grey corresponds to the area shown in the main view. To the left of the tree the temporal chunks are shown; the list of nodes-types, to the right of the tree, alters the nodes that are shown on the gestalt view.

Figure 6.6: Two uses of gestalt views to provide overview plus context displays: the game Sid Myer's Alpha Centauri, and a prototype view in Swaca.

ble tree. One solution to the 'weekend problem' would be to show the chunk with the most recently created actions, if the "today and yesterday" chunk did not have any actions in it.

There are a number of improvements that could be made to Swaca to provide a better user experience.

- More features for the editor, including goto line, find and replace, and syntax highlighting.

- Exporting and importing documents to and from the file-system.

- One participant suggested that the names of landmarks should be propagated to the *start* of the branch, so the user can scan up the left-hand side of the tree and see the names of many of the documents. The names could also be derived from file-names used in imports and exports, and the subject-lines from documents sent by email.

- Bugs in drawing and dragging the tree-cursor need to be fixed. The current cursor-movement algorithm uses a brute-force search to determine the point in the tree that is closest to the mouse — so the tree-marker can 'snap' to the nearest node. However, the search is slow when the tree becomes large; changing the algorithm, so it uses a kd-tree for example, would make interface more fluid.

- Finally, adding the ability to search the tree, as discussed in Section 6.1.2.4, should ease document management with the tree. As a supplement to searching, user-defined categories could be added to landmarks, so only branches that match a particular category are shown.

Contrary to expectations, none of the participants asked for the ability to delete nodes to be added to Swaca.

6.3 Conclusion

In this chapter I have discussed the creation and evaluation of a temporal document-organisation system that did not rely on files. While usability

problems were discovered, in general participants found that the system was useful, usable, and did not confuse. In response to the usability problems, I have proposed some solutions that should make the timeline-based system easer to use.

There are a number of guidelines that can be drawn from this study of Swaca.

- Implementing landmarks should be considered when creating a visualisation of undo, or similar error-recovery systems, as landmarks were liked by the participants, and comments indicated they were useful in finding prior document states.

- From the participants' comments, it should be expected that a timeline will be used to correct large errors; the design of the interface should multiple actions easy to undo to take this into account.

- The participants in the evaluation of Swaca have shown that it is *possible* to use a branching timeline as a document organisation system. While it is unknown if users *will* use a timeline to create documents if they also have the ability to create files, the possibility does exist, so the system should be designed with this in mind.

- Zooming the timeline does not appear to be an effective navigation technique. A gestalt view of the timeline may prove to be a useful and usable way to navigate the timeline.

- It appears that icons, which implicitly mark important states, are an effective way to remind the user of past tasks. The designer of a system will have to determine which actions are important, as they are likely to differ between document types.

Swaca was created and evaluated in order to gain insight into the last of my three research questions: is it possible to combine document and version retrieval in the same interface, and will it be useful? I showed that it is indeed possible to combine version and document retrieval in the same interface, and that the system is useful and usable. However, the results of the evaluation

presented in this chapter can only be considered as a partial answer to my research question for three main reasons. First, it is unknown if the Swaca system is more useful and usable than other document-organisation systems. Second, there may be better ways of combining documents and versions, which I have not considered. Finally, it is unknown if the timeline-based method of organising documents and versions would be effective outside the text-editing domain.

Chapter VII

Conclusion

This thesis presented a study of temporal document-organisation and version organisation. In it, I discussed how time is a compelling way to organise documents and versions: it is an effective cue to retrieval, temporally ordered documents act as reminders of current activities, temporal ordering can be applied to all system objects, and documents and versions can be organised automatically. I used theoretical and empirical methods to assess a number of temporal systems; from the evaluations, I formed a number of guidelines for the creation of temporal and non-temporal systems. These were then used for the creation of a system that organised documents without the need for names or folders.

My study began in Chapter 2, where I looked at the human factors that affected temporal document-organisation and version organisation. I discussed how the information-storage model became accepted because it explained the filled-duration illusion. I then examined how human memory in general, and autobiographical memory in particular, affect temporal document organisation. An important finding from prior research is that people are good at remembering the order of events, such as creating or editing a document, but quite poor at remembering dates. I then looked at the user's tasks with documents: finding documents as part of information foraging, and being reminded of current activities. The user's tasks with versions were then examined, and I showed how there are two main types of error-recovery action that can be carried out: linear and branched error-recovery.

A user-centred study of non-temporal and temporal document-organisation and version organisation systems was then presented in Chapter 3. After introducing some cues to document retrieval, I showed how some non-temporal document organisation systems supported the user's tasks with

documents and versions, which were introduced in Chapter 2. I looked at how time is exploited in temporal document-organisation systems in order to support the user's tasks with documents. I showed that some systems, such as the back button found in Web browsers, provide few cues to document retrieval, but are commonly used. Others, such as recent files lists, should be good at reminding the user of the current tasks, and allow recently edited documents to be found easily. I also discussed some experimental document-organisation systems, such LifeStreams. It seems that many temporal document-organisation systems do *not* supply dates as a cue to document retrieval, because people have a poor memory for dates. I also showed that titles seem to be an important cue to document retrieval, and that clusters of related documents (or 'episodes') are found in many interfaces.

I then looked at version retrieval systems (Section 3.3). Such systems store multiple versions of documents, which are then retrieved as part of error-recovery or system exploration. For example, VMS and the WikiMedia system create a new version of a file when the user saves, while undo creates a version whenever the user carries out an action that modifies the document. I discussed the many types of undo, and how the common stack-based undo has the same problem as the back button in Web browsers: it can lose state. Other systems that I studied, such as history undo, do not lose state, but they may be more confusing than stack-based undo. I also looked at version retrieval systems that do not store data in a linear data structure, but use a tree to organise versions.

Chapters 4 and 5 presented the first empirical evaluations of some of the most common temporal document-organisation and version-organisation systems. The research goal of the first evaluation was to partly determine the 'best' way to organise documents by time. This was done by looking at the retrieval of documents from history lists, which are found in many applications. I assessed four different types of list in a controlled experiment, and discovered that systems should split the history into days: it did not help or hinder retrieval, but it is preferred by users. However, breaking up the list using a non-temporal classification scheme, as is done by Microsoft Internet Explorer, could either help or hinder retrieval. If the user knew the exact title of the document being sought, but not the exact date, then retrieval

was faster using classified list; in all other cases retrieval was faster from the systems that did not classify documents.

I then conducted a two-part evaluation of error-recovery, using keystroke-level analysis and an empirical evaluation. The aim was to gain some insight into the most effective way to use time to organise different versions of the same document. The theoretical analysis shows that, for an expert using a text editor, forward error-correction is the fastest recovery method if the number of errors is fewer than six. However, a visualisation of a tree-based error-recovery method allows for faster recovery from larger and more complex errors, such as a branched error-recovery task. The empirical evaluation looked at error-recovery in two different editing domains: text and drawing. The results confirmed that, for the text editor, forward error-correction is the optimal method for correcting simple errors; undo is the optimal method of recovering from simple errors in the drawing domain. However, a tree-based visualisation is the optimal method of recovering from complex errors in both editing domains. I also showed that people tended to use the tree-based error-recovery method to recover from large errors in an open text-editing task.

I then presented the evaluation of a text editor, SWACA, which only used time to organise documents and document versions (Chapter 6). SWACA was created to see if combining temporal document and version organisation in the same interface is possible; it was empirically evaluated to determine whether the resulting system was useful and usable. The design of SWACA was informed by the results and guidelines presented in the prior chapters. Unlike most other temporal document organisation systems, SWACA does not supplement file systems, but replaces files entirely. Retrieval was carried out using an interface that was similar to the tree-based error-recovery method, but included features that should make the system more useful and usable: icons implicitly mark 'significant actions', landmarks allow the user to explicitly mark important states, temporal chunks break the timeline into manageable parts, and zooming increases or decreases the amount of information shown to the user. In the usability study I found that the system was both useful and usable. Participants could find documents, and tell them apart from document versions. I found that the tree was typically used

to correct larger errors; while there were problems with zooming and the temporal chunks, the icons and landmarks were liked, and helped retrieval. However, the question of whether temporal document-organisation is better than a file systems was not answered by this evaluation.

7.1 The Future

In this section I will discuss future work and future developments in three areas: history lists (Section 7.1.1), tree-based error-recovery mechanisms (Section 7.1.2), and temporal document-organisation (Section 7.1.3).

7.1.1 History Lists

While an initial answer to my research question was found, the best way to organise documents by time was not established in the evaluation of history list interfaces (Chapter 4). I see two clear areas for future work in history lists that lead on from my prior research, beyond testing other temporal document-organisation interfaces and the retrieval of other document types.

In the first, work is needed to establish the performance of the systems when the participant has an prior knowledge of the history. The participants in the evaluation of history lists in Chapter 4, and the evaluation of the Outride system (Pitkow et al., 2002), were provided with an artificial history that they had no prior knowledge of. This was not optimal, as a participant should be able to retrieve documents from a history list faster if he or she knows what is in the list. In addition, episodes should be present in a true history, which would also speed retrieval as the participant could search for clusters of related documents, rather than individual documents. A study that took advantage of the participant's prior knowledge of the history would allow many of the benefits of temporal document-organisation to be quantified for the first time.

I also see an opportunity for a study that looks at the optimal temporal-chunking method. Participants in the history-list evaluation (Chapter 4) preferred the chunking interface, but only one chunking-method was evaluated: days. However, other chunking mechanisms are possible, such as that employed by Microsoft Internet Explorer, which uses a mixture of days and

weeks, or SWACA, which had chunks based on linguistic norms (such as "today", "yesterday" and "this week"). A study of document retrieval patterns would help establish the optimal chunking method for a particular document type. I suspect that the users' interaction patterns would be quite different for different document types, so the study would need to be able to be generalised so it can be applied to a range of interfaces.

7.1.2 Tree-Based Error-Recovery

My research into temporal organisation of document versions has provided initial insight into the question of which interface is the best, with the tree-based undo visualisation proving to be the most effective overall. However, not all error recovery methods, or all document editing types, were evaluated in Chapter 5. Further work is needed to establish the efficacy of other error-recovery interfaces.

The tree-based undo visualisation was one of the more radical interfaces that I evaluated in this thesis. Despite being unusual, I predict that many interfaces could adopt the tree in the future. By using the tree, users should be able to correct large and complex errors more quickly and easily than is possible with other error-recovery methods. Existing systems could be easily adapted to work with the tree: the data-model for tree-based undo is very similar to that currently used for stack-based undo, and as shown in SWACA, no change to the traditional stack-based undo semantic is needed.

Icons, used on the timeline in SWACA, are similar to the icons and action descriptions that are found in the undo visualisations of many existing systems, such as Microsoft Word, Adobe Photoshop, and the GIMP. Therefore, such systems should be well suited to adding icons to any tree-based error-recovery system. Landmarks should also be useful. Adobe Photoshop already has 'snapshots', which are similar to landmarks, so system such as this should be able to implement landmarks without too much difficulty.

When adapted to a new system, the timeline would need to be assessed, particularly if it was adapted to an untested editing domain. In addition further research is needed to determine the ideal method of managing a large undo tree. Zooming was assessed in the prior chapter, but this turned out

to be less than ideal. However, a gestalt view, or even a fish-eye lens effect, may be useful.

I investigated the use of the tree-based error-recovery system in a single-user environment. Research is needed to see if real-time collaboration is possible using a tree-based error-recovery method in a computer-supported collaborative workspace. Such as system would share the problems of existing collaborative undo systems, but existing CSCW solutions may be able to be used.

I suspect that if a timeline is provided for error-recovery, then people will begin to use it as a document-organisation method. Research is be needed to see if this is indeed the case.

7.1.3 Temporal Document-Organisation with Swaca

The creation of Swaca (Chapter 6) has shown that document and version retrieval is able to be combined, and that the resulting system is useful and usable. However, my study did not show that documents and versions can be found more quickly and easily in Swaca than in any other document organisation system, or that the documents and versions in the system were better at reminding the user of his or her current tasks, compared with other document organisation systems.

There are a number of open research areas relating to temporal document organisation using the Swaca system. One is the evaluation of systems to support document retrieval, and version retrieval, when the tree becomes large. The temporal chunking mechanism in Swaca was designed to support this task by only showing recently edited branches, because old information is generally less useful (Section 2.3.1). However, other techniques may be needed to supplement browsing the tree in the rare occasions that old documents, or document versions, are needed.

- One technique, mentioned in Section 6.1.2.4, is to search the tree for some text. Other temporal document-organisation systems have implemented searching, such as LifeStreams (Freeman, 1997), as have many non-temporal systems, such as standard file-systems found on desktops today. Indeed, searching the file system on the Xerox Star was seen as

necessary, as user's "categorisation schemes are frequently ad hoc and idiosyncratic" (Smith et al., 1982).

- As mentioned in Section 6.2.6, a gestalt view of the timeline should allow the user to get an overview of the timeline while also being able to see the detail. The gestalt view in the TimeSpace temporal document-organisation system was liked by the participants in the observational study conducted by Krishnan and Jones (2005), but testing is needed to determine if a gestalt view would allow for fast and easy retrieval of old documents.

- Many participants in the study of SWACA asked for the ability to add metadata, such as classifications, to the timeline. I imagine that categories in SWACA would allow the user to associate a keyword or label to a node, and then view all nodes that match a category. Users may also be able to associate categories with particular 'activities', allowing a hybrid temporal and activity based document organisation, much like the TimeSpace system (Krishnan and Jones, 2005).

- A final method for assisting the user with finding old documents is to add *more* data to the timeline, in order to provide more cues to document retrieval. Examples of information that could be added include other documents that the user viewed (such as Web pages and email messages) and the location of the user and others in the area, relying on systems such as the Active Badge (Harter and Hopper, 1994; Harter et al., 2002; Schilt et al., 2002).

While old documents may be hard to find in a purely temporal system, but it is unknown if it would be any harder than locating old documents in existing systems.

Another area that is requires research is how to integrate a purely temporal system, such as SWACA with the wider computing environment. Emailing and printing both export data from SWACA, and this could be extended by allowing the user to drag actions from the timeline and drop them into a folder, which would create a file that would contain the same contents as the

document represented by the node. Getting data *into* the purely temporal system could work the same way, with a dropped document added to the tree as a new branch.

Swaca only provided access to plain-text documents, but a fully-featured timeline would need to work with multiple document types. One solution, which will need assessment, is to change 'modes' when the user retrieves a branch that represents a document of a different type: changing from a spreadsheet to a text-editor for example.

At some stage users of temporal document-organisation systems may want to write documents in collaboration with others. One source for ideas for how this may be done is in the area of asynchronous collaboration, where temporal systems such as Track Changes in Microsoft Word (Microsoft, 1998), Cvs (Fogel, 1999) and Subversion (Collins-Sussman et al., 2004) allow the user to accept, or reject, changes made to a document by another user. However, the best way to visualise the process of receiving, and accepting, the changes is unknown.

Is it possible to create a document organisation system that does not have the drawbacks of traditional file-systems? I believe it is, but the little empirical evidence I have is circumstantial. However, it should be possible to create an evaluation that would allow this question to be answered.

I do not believe that time is a panacea for document-organisation, but temporal document-organisation is an excellent solution to many of the user's problems in everyday computing.

Appendix A

Swaca Manual

The manual for Swaca was written after the paper-prototype (Section 6.1) but before the system was completed, as recommended by Norman (1990). As Swaca was developed, the manual was modified accordingly. The following manual formed the on-line help that was provided in the version of Swaca that was used in the evaluation (Section 6.2), with the screen-shots representing the final state of the windows that were displayed as part of Swaca.

Swaca Manual

Michael JasonSmith <mpj17@student.canterbury.ac.nz>

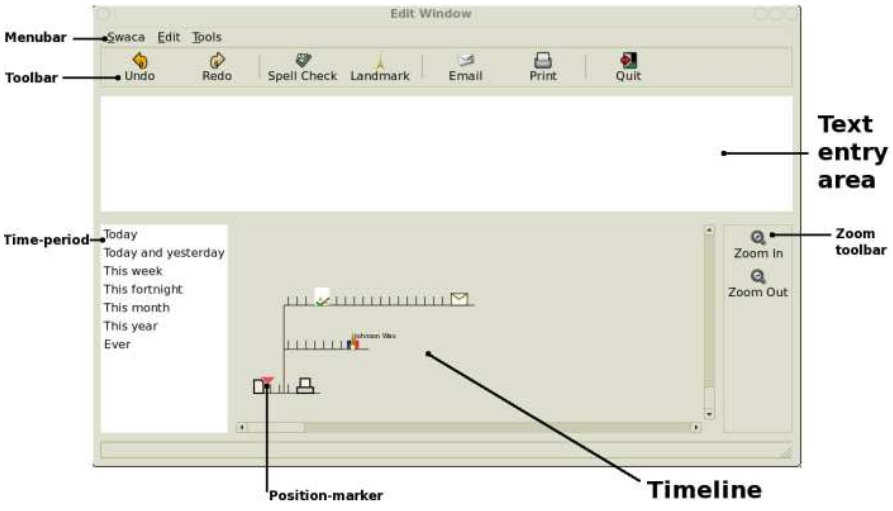
Table of Contents

1. Introduction

Swaca is a text editor that has an integrated undo and versioning system that eliminates the need for files. There are two main parts to the Swaca: the text entry area, and a timeline that shows the actions you have performed. One of the unique features of Swaca is that documents are represented as branches on the timeline, rather than files in folders. This manual will show you how to edit text and use the timeline.

To start Swaca type **swaca** at the command-line, and the main Swaca window will open.

Figure 1. The Main Swaca Window

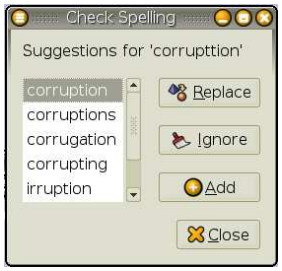


2. Editing Text

For the most part, editing text in Swaca is the same as editing text in a normal text editor: the cursor-control keys moves the text-entry point, holding **Shift** while moving the cursor selects text, and cut, copy and paste are bound to the same keys and have the same behaviour as most other editors (excluding vi and Emacs). Besides the simple text-editing tasks, Swaca can check spelling, print the text, and email the text. As you edit the text, or perform any of these other actions, icons that represent the completed actions are placed on the timeline (see Section 3, "Using the Timeline").

2.1. Checking Spelling

Figure 2. Spellcheck Dialog



To check the spelling of the text:

1. Click Spellcheck in the toolbar.
2. For each word that is not found in the dictionary, the Spellcheck dialog (Figure 2, "Spellcheck Dialog") will appear with a list of suggested replacements. From this window, you can

   • Replace the incorrect word with a correct word,

   • Ignore the word, or

   • Add the word to the dictionary.

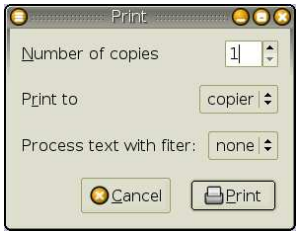
   To cancel the spell-check, close the Spellcheck dialog.

Personal Dictionary

Swaca uses the standard aspell system to check the spelling of the document, so the personal dictionary that is used (`~/.aspell.en.pws`) should be the same as that used by many applications.

2.2. Printing Text

Figure 3. Print Dialog



To print the text:

1. Click Print in the toolbar. This will open the Print dialog (Figure 3, “Print Dialog”).
2. Select the number of copies you wish to print, and the printer you wish to print to.
3. Select the filter used to process the document before printing.
4. Finally, click Print.

By default there are three filters provided with Swaca.

a2ps
: This filter passes the text through a2ps so it is printed *two-up*.

LaTeX
: This filter passes the text through LaTeX and dvips before printing it.

none
: This filter does not alter the text in any way, and simply prints it using lpr.

For more information on the printer filters, see Appendix A, *Printer Filters*

2.3. Emailing Text

Figure 4. Email Window

To email the currently displayed text:

1. Click Email, this will open the Email dialog (Figure 4, “Email Window”),
2. Fill out the fields in the Email Window (the To field is compulsory), and
3. Click Email.

Currently Swaca has no mechanism for recieving email.

SMTP Server

Swaca assumes that localhost runs an SMTP server.

3. Using the Timeline

The timeline, located beneath the text-entry area, is used to correct errors and manage documents. As you perform actions in the editor, icons are placed on the timeline, which grows from left to right. There are six icons shown on the timeline:

- The text-editing icon

that represents all simple text-editing tasks, such as typing, deleting or copying text;

- The pause icon

that is created after no editing task has been undertaken for ten minutes;

- The spell-check icon

that is created after you spell-check the document;

- The print icon

that is created whenever you print the document;

- The email icon

that is created when you email the document; and

- The landmark icon

that you create explicitly when required (see Section 3.5, “Managing Landmarks”).

Clicking on any of these icons with the mouse will allow you to view information about the node. A red marker

indicates the current position on the timeline. Moving this marker allows you to correct errors, create documents, and find documents.

3.1. Correcting Errors

To correct an error using the timeline either:

- Click Undo (**Control**-**z**), or
- Drag the timeline marker to a point where the error does not exist.

Using Undo moves the timeline marker left, to a point to just *before* the undone action was performed, which is equivalent to dragging the marker to the same position. When you resume editing a new branch is created on the timeline.

Actions that Cannot be Undone

Printing and emailing actions cannot be undone. While icons representing these actions appear on the timeline, moving the timeline marker to the left of a printing or emailing icon will have no effect.

To redo an action (to undo an undo) either:

- Click Redo (**Control**-**y**), or
- Drag the timeline marker to the state where the undo was started from.

If there are multiple branches from a point, Redo will select the most recently created branch, which emulates the behaviour of redo in programs such as Microsoft Word and Adobe Photoshop.

3.2. Creating a New Document

To create a new document

1. Drag the timeline marker to just to the *right* of the *root* document state at the bottom left of the timeline, which looks like

, and

2. Start editing from the blank document state.

This will create a new branch in the timeline, representing your new document.

Alternatively, you can find a location on a branch that closely matches the initial structure of the document that you wish to create, such as the start of a letter. Continuing editing from that point will create a branch that corresponds to the new document.

3.3. Finding a Document

Finding a document in Swaca requires you to find the branch that represents the document, usually by looking for particular features on the timeline, such as pause, spell-check, or printing icons. Normally this is quite simple as you will be looking for recently created icons. Two tools are provided to further assist you in finding documents: you can reduce the detail shown on the timeline, and you can alter the number of actions shown on the timeline.

3.3.1. Reducing Detail

To reduce amount of detail shown on the timeline (and reduce the size of the icons) click Zoom Out, which is to the right of the timeline. The actions that are shown at each zoom-level are shown in Table 1, "The Actions that are Shown at Different Zoom Levels". You may need to reduce the detail because important actions may be hard to find as the timeline displays icons for all the actions that have been performed (by default).

To increase the amount of detail shown click Zoom In.

Table 1. The Actions that are Shown at Different Zoom Levels

Level	Edit	Pause	Spell	Print	Email	Landmark
4 (Default)	✔	✔	✔	✔	✔	✔
3	✘	✔	✔	✔	✔	✔
2	✘	✘	✘	✔	✔	✔
1	✘	✘	✘	✘	✘	✔

The branches with no icons on them are not shown by Swaca. For example, only branches with landmarks on them are shown at zoom-level 1. However, the branch with the marker on it is always shown.

3.3.2. Altering the Number of Actions Shown

The number of actions shown on the timeline can be altered by selecting an entry from the Time Period list to the left of the timeline. All time periods represent time-spans *relative* to the current time. The time periods are as follows.

Today
: All actions that occurred in the last 24 hours.

Today and yesterday
: All actions that occurred in the last 48 hours (which is the default).

This week
: All actions that occurred in the last 7 days.

This fortnight
: All actions that occurred in the last 14 days.

This month
: All actions that occurred in the last 28 days.

This year
: All actions that occurred in the last year.

Ever
: All actions that have occurred.

Time Calculations

All times periods are calculated using seconds, and the tropical year (31,556,925.187s) is used rather than the sidereal, lunar, Julian, Gregorian. . . .

3.4. Viewing Icon Information

Figure 5. Print Information Window

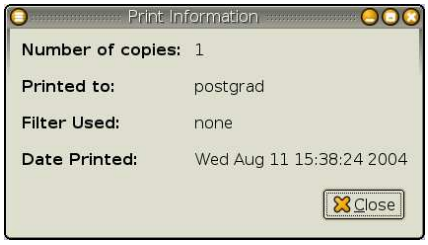
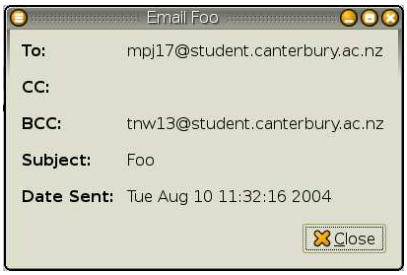


Figure 6. Email Information Window



To view information about the action that created the icon, click on the icon; a dialog that shows the relevant information will appear. The information shown differs from icon to icon. For example, the print information window (Figure 5, “Print Information Window”) differs quite a lot from the email information window (Figure 6, “Email Information Window”). The only datum that is consistently displayed in all the information windows is the date that the action was carried out.

3.5. Managing Landmarks

Landmarks are a way to mark a locations on the timeline that you consider important so you can quickly and easily find them. They are similar in propose to bookmarks in Web browsers, which allow you to quickly and easily return to Web pages. In this section we will look at how to create, edit, and delete landmarks.

3.5.1. Creating Landmarks

Figure 7. Landmark Properties Window

To create a landmark:

1. Move the timeline-cursor to the location on the timeline where you want the landmark to appear.
2. Click Landmark (**Control**-**l**) and a landmark icon will appear on the timeline and a Landmark Properties window will appear (Figure 7, “Landmark Properties Window”).
3. Give the landmark a name. The name will be displayed next to the landmark icon on the timeline. Names do not have to be unique, but unique names should be easier to find.
4. If desired, add a note to the landmark before closing the window.

3.5.2. Editing Landmarks

To edit a landmark locate the landmark and double-click on the landmark icon, and the Landmark Proprieties window (Figure 7, “Landmark Properties Window”) will open with the relevant details. When you have finished editing the landmark, click Close.

3.5.3. Deleting Landmarks

To delete a landmark, open the landmark property window and click Delete. Note that landmarks are unique in Swaca as they are the only node that can be deleted.

Printer Filters

Swaca can use a filter to process text before it is printed. A filter is an executable that takes two arguments: the name of the printer to print to, and the title of the print-job. The filter reads text from standard-input and sends the processed document to the printer (usually using lpr). There are three filters provided by default.

none.filter
: Does nothing other than spool the text with lpr.

a2ps.filter
: Processes the text with a2ps so it is printed *two-up*.

latex.filter
: Processes the text with LaTeX and dvips before spooling the document with lpr.

User-written filters can be added to `~/.swaca/printer_filters/`. It should be noted that the `.filter` extension is compulsory.

Appendix B

Swaca Evaluation Structured Interview

This structured interview was used for the evaluation of the Swaca text-editor (Section 6.2). The first group of questions focused on the participant's background, particularly their experience with editors (Section B.1). Then, in Section B.2, I asked questions about the participant's general use of Swaca. The next three blocks of questions looked at document retrieval, error recovery and document storage in Swaca (Sections B.3–B.5). The interview concluded with some questions about Swaca that did not fit into the other categories (Section B.6)

B.1 Background

1. What are your normal document editors?

   (a) What do you use them for?

(b) Rate how often you use the editors on the scale

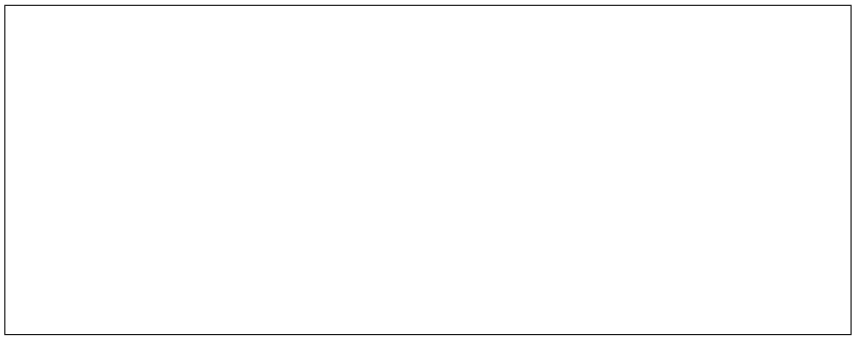
Never	1	2	3	4	5	Always

2. How do you activate undo in your normal editor?

Keyboard	Mouse	Never

3. What is your normal method of opening files?

- What don't you like about it?



- What do you like about it?

4. Rate your use of the following techniques for opening files.

   - Open

Never	1	2	3	4	5	Always

     What applications and why?

   - Recent files

Never	1	2	3	4	5	Always

     What applications (or documents) and why?

   - File browser and desktop

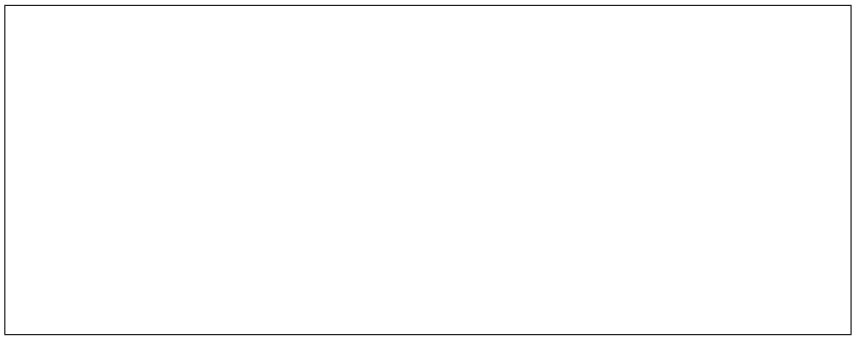
Never	1	2	3	4	5	Always

     What documents and why?

- Command-line

Never	1	2	3	4	5	Always

What documents and why?



5. I normally start documents from scratch, rather than basing them on an older document.

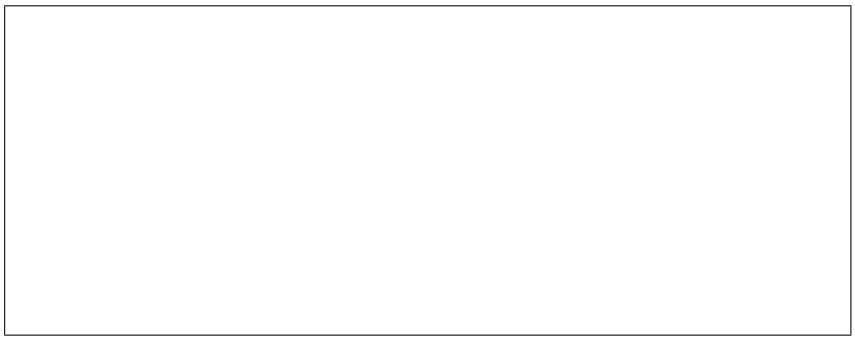
Disagree	1	2	3	4	5	Agree

- Why?

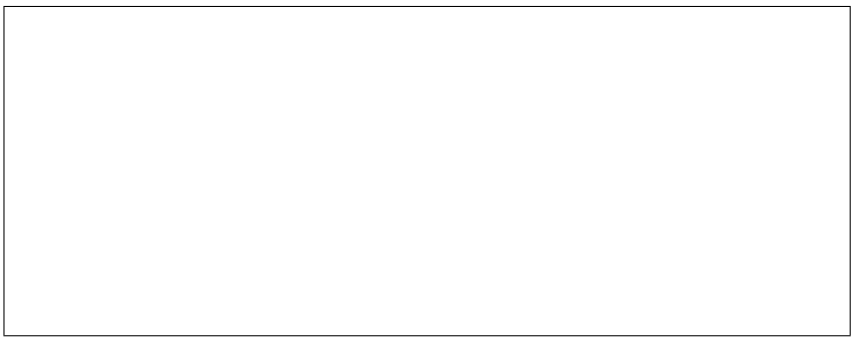
B.2 General Use of Swaca

1. What did you use it for?

- Why?



2. What other editors did you use at the same time?



- Why?

3. Were the documents you created in Swaca larger or smaller than normal?

Smaller	1	2	3	4	5	Larger

   - Why?

4. Are you still using Swaca?

Never	1	2	3	4	5	Always

   - Why?

B.3 Document Retrieval in Swaca

1. It was easy to find documents using Swaca.

Disagree	1	2	3	4	5	Agree

   - What was difficult about finding documents?

2. Zooming helped finding documents in Swaca.

Disagree	1	2	3	4	5	Agree

   - Why?

3. The email, print and spell check icons on the timeline helped finding documents in Swaca.

Disagree	1	2	3	4	5	Agree

   - Why?

4. The time-chunks helped finding documents in Swaca.

Disagree	1	2	3	4	5	Agree

- Why?

5. Landmarks helped finding documents in Swaca.

Disagree	1	2	3	4	5	Agree

- Why?

6. Swaca was better than your normal method at finding documents.

Disagree	1	2	3	4	5	Agree

- Why?

B.4 Error Correction in Swaca

1. Did you use "normal undo" (C-z, C-y) to correct errors much?

Never	1	2	3	4	5	Always

   - What sort of errors did you correct using normal undo?

   - Why?

2. Did you use the tree to correct errors much?

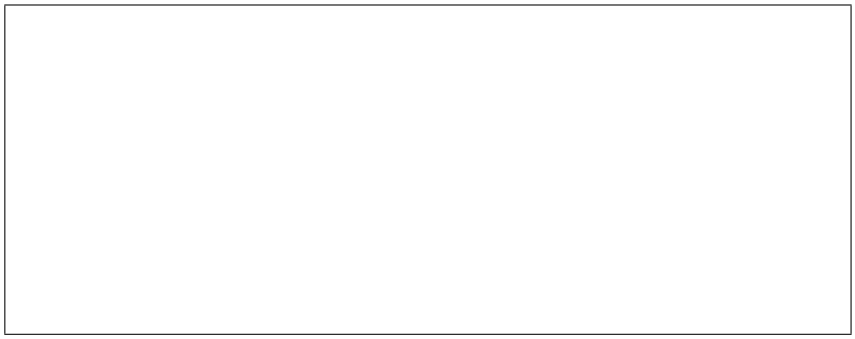
Always	1	2	3	4	5	Never

   - How did you use the tree?

Double click	Drag

   - What sort to errors did you correct using the tree?

- Why?



3. The tree was better than undo.

Disagree	1	2	3	4	5	Agree

B.5 Storage and Swaca

1. I wanted to invoke save while using Swaca.

Disagree	1	2	3	4	5	Agree

2. Landmarks were better than named files.

Disagree	1	2	3	4	5	Agree

3. I was confident that data would not be lost.

Disagree	1	2	3	4	5	Agree

B.6 Other Questions about Swaca

1. The lack of files was confusing.

Disagree	1	2	3	4	5	Agree

2. I could readily distinguish between ‘corrections’ and ‘documents’.

Disagree	1	2	3	4	5	Agree

3. I normally started new documents from scratch, rather than basing them on an older document.

Disagree	1	2	3	4	5	Agree

   - Why?

4. What would you change about Swaca, if you could?

Bibliography

Abowd, G. and Dix, A. (1995). Giving undo attention. *Interacting with Computers*, 4(3):317–342.

Abowd, G. D. and Mynatt, E. D. (2000). Charting past, present, and future research in ubiquitous computing. Acm *Transactions on Computer-Human Interaction*, 7(1):29–58.

Abrams, D., Baecker, R., and Chignell, M. (1998). Information archiving with bookmarks: Personal Web space construction and organization. In Karat et al. (1998), pages 41–48.

Abu-Shakra, M. and Fisher, G. L. (1998). Multi-grain version control in the Historian system. In Magnusson, B., editor, *System Configuration Management:* Ecoop'98 Scm-9 *Symposium*, volume 1439 of *Lecture Notes in Computer Science*, pages 46–56, Brussles, Belgium. Springer, Berlin, Germany.

Adobe (2002). *Adobe Photoshop User Guide*. Adobe Systems Incorporated, San Jose, California. Corresponds to version 7.0 of Adobe Photoshop for Windows and Macintosh.

Anderson, C. J. (2005). Calendar and reverse calendar effects: Time peaks in memory as a function of temporal cues. *Memory*, 13(2):113–123.

Apple (2001). *Aqua Human Interface Guidelines*. Apple Computer, Incorporated, Cupertino, California.

Archer, Jr., J. E., Conway, R., and Schneider, F. B. (1984). User recovery and reversal in interactive systems. Acm *Transactions on Programming Languages and Systems*, 6(1):1–19.

Arnold, B., van der Veer, G., and White, T., editors (1993). *Proceedings of the* Chi *'93 Conference on Human Factors in Computing Systems*, Amsterdam, The Netherlands. Ngi, Ieee-cs, Ifip, Acm Sigcaph, Siggraph, Siggroup and Sigchi, Acm Press, New York, New York.

Association for Computing Machinery (2004). The Acm digital library. Web page. Available from `http://portal.acm.org/dl.cfm`.

Augustine of Hippo (1993). *Confessions*. The Folio Society, London, England. First published ad 397. Original translation by J. G. Pilkington, published in 1876 by T. & T. Clark, Edinburgh, Scotland.

Aula, A., Jhaveri, N., and Käki, M. (2005). Information search and re-access strategies of experienced Web users. In Ellis, A. and Hagino, T., editors, *Proceedings of the 14th international conference on World Wide Web*, pages 583–592, Chiba, Japan. Acm, Acm Press, New York, New York.

Baecker, R. M., Nastos, D., Posner, I. R., and Mawby, K. L. (1993). The user-centered iterative design of collaborative writing software. In Arnold et al. (1993), pages 399–405.

Bardram, J. E. (2004). Applications of context-aware computing in hospital work: examples and design principles. In Sac'04: *Proceedings of the 2004* Acm *symposium onApplied computing*, pages 1574–1579, New York, New York. ACM Press.

Barreau, D. and Nardi, B. A. (1995). Finding and reminding: File organization from the desktop. Acm Sigchi *Bulletin*, 27(3):39–43.

Beckett, D. (2002). *Expressing Simple Dublin Core in* Rdf/Xml. Dublin Core Metadata Initiative. Available from `http://dublincore.org/documents/2002/07/31/dcmes-xml/`.

Beckett, D. (2004). RDF/XML *Syntax Specification (Revised)*. World Wide Web Consortium, Cambridge, Massachusetts. Available from `http://www.w3.org/TR/rdf-syntax-grammar/`.

Bederson, B. B. (2001). PhotoMesa: A zoomable image browser using quantum treemaps and bubblemaps. In Marks and Mynatt (2001), pages 71–80. Available from `http://www.cs.umd.edu/hcil/photomesa/`.

Benson, C., Elman, A., Nickell, S., and Robertson, C. Z. (2004). *GNOME Human Interface Guidelines 2.0*. The GNOME Usability Project, second edition. Available from `http://developer.gnome.org/projects/gup/hig/2.0/`.

Berners-Lee, T., Fielding, R., and Masinter, L. (1998). *Uniform Resource Identifiers (URI): Generic Syntax*. The Internet Society. RFC 2396, available from `http://www.ietf.org/rfc/rfc2396.txt`.

Blair, D. C. and Maron, M. E. (1985). An evaluation of retrieval effectiveness for a full-text document-retrieval system. *Communications of the* ACM, 28(3):289–299.

Brickley, D. and Miller, L. (2005). FOAF *Vocabulary Specification*, 'Pages about Things' edition. Available from `http://xmlns.com/foaf/0.1/`.

Brin, S. and Page, L. (1998). The anatomy of a large-scale hypertextual web search engine. *Computer Networks and* ISDN *Systems*, 30:107–111. Available from `http://www7.scu.edu.au/programme/fullpapers/1921/com1921.htm`.

Brown, J., Graham, T. C. N., and Wright, T. (1998). The Vista environment for the coevolutionary design of user interfaces. In Karat et al. (1998), pages 376–383.

Brown, S. W. (1995). Time, change, and motion: The effects of stimulus movement on temporal perception. *Perception and Psychophysics*, 57(1):105–116.

Bush, V. (1945). As we may think. *Atlantic Monthly*, 176(1):101–108.

Card, S. K. and Austin Henderson, J. (1987). A multiple, virtual-workspace interface to support user task switching. In Carroll, J. M. and Tanner, P. P., editors, *Proceedings of the* SIGCHI/GI *conference on Human factors in computing systems and graphics interface*, pages 53–59, Toronto, Ontario. ACM SIGCHI, ACM Press, New York, New York.

Card, S. K., Moran, T. P., and Newell, A. (1983). *The Psychology of Human-Computer Interaction*. Lawrence Erlbaum Associates, Hillsdale, New Jersey.

Card, S. K., Robertson, G. G., and Mackinlay, J. D. (1991). The information visualiser, an information workspace. In Robertson, S. P., Olson, G. M., and Olson, J. S., editors, *CHI '91 Conference Proceedings*, pages 181–188, Louisiana. ACM SIGCHI, ACM Press, New York, New York.

Carpendale, M. S. T. and Montagnese, C. (2001). A framework for unifying presentation space. In Marks and Mynatt (2001), pages 61–70.

Carroll, J. M., Neale, D. C., Isenhour, P. L., Rosson, M. B., and McCrickard, D. S. (2003). Notification and awareness: synchronizing task-oriented collaborative activity. *International Journal of Human-Computer Studies*, 58(5):605–632.

Carroll, L. (1997). *Alice's Adventures in Wonderland*. Project Gutenberg. Available from `http://www.ibiblio.org/gutenberg/etext97/alice30h.htm`.

Catledge, L. D. and Pitkow, J. E. (1995). Characterizing browsing strategies in the World-Wide Web. In *Computer Systems and* ISDN *Systems: Proceedings of the Third International World Wide Web Conference.*, volume 27, pages 1065–1073, Darmstadt, Germany.

Clasen, M. (2005). *GTK+ Reference Manual*. GTK+ team. Available from `http://gtk.org/api/`.

Cockburn, A. and Jones, S. (1996). Which way now? Analysing and easing inadequacies in WWW navigation. *International Journal of Human-Computer Studies*, 45(1):105–129.

Cockburn, A. and McKenzie, B. (2001). What do web users do? An emperical analysis of web use. *International Journal of Human-Computer Studies*, 54(6):903–992.

Cockburn, A., McKenzie, B., and JasonSmith, M. (2002). Pushing back: Evaluating a new behaviour for the back and forward buttons in web browsers. *International Journal of Human-Computer Studies*, 57(5):397–414. Avaliable from `http://www.cosc.canterbury.ac.nz/andrew.cockburn/papers/backEval.pdf`.

Collins-Sussman, B., Fitzpatrick, B. W., and Pilato, C. M. (2004). *Version Control with Subversion*. O'Reilly & Associates, Sebastopol, California, first edition. Available from `http://svnbook.red-bean.com/`.

Compaq (1999). *OpenVMS User's Manual*. Compaq Computer Corporation, Houston, Texas.

Conn, A. P. (1995). Time affordances. In Irvin R., K., Mack, R., and Marks, L., editors, *Proceedings of the* CHI *'95 Conference on Human Factors in Computing Systems*, pages 186–193, Denver, Colorado. ACM SIGCHI, ACM Press, New York, New York.

Cooper, A. and Reimann, R. (2003). *About Face 2.0: The Essentials of Interaction Design*. Wiley Publishing Inc., Indianapolis, Indiana, second edition.

Corbató, F. J., Merwin-Daggett, M., and Daley, R. C. (1962). An experimental time-sharing system. In AFIPS *Conference Proceedings, Spring Joint Computer Conference*, volume 21, pages 335–344, San Francisco, California. Available from `http://larch-www.lcs.mit.edu:8001/~corbato/sjcc62/`.

Coschurba, P., Baumann, J., Kubach, U., and Leonhardi, A. (2001). Metaphors and contex-aware information access. *Personal and Ubiquitous Computing*, 5(1):16–19.

Creative Commons (2004). RDF-enhanced search. Web page. Available from `http://search.creativecommons.org/`.

Creative Commons (2005). *Implementing Creative Commons Metadata*. Creative Commons, San Francisco, California. Available from `http://creativecommons.org/technology/metadata/implement`.

Cunningham, S. J. and Connaway, L. S. (1996). Information searching preferences and practices of computer science researchers. In *Proceedings of the Sixth Australian Conference on Computer-Human Interaction*, pages 294–299, Hamilton, New Zealand. IEEE Press, New York, New York.

Czerwinski, M. and Horvitz, E. (2002). Memory for daily computing events. In Faulkner, X., Finlay, J., and Detienne, F., editors, *People and Computers* XVI*: Memorable Yet Invisible*, pages 230–245, London, United Kingdom. British Computer Society, Springer-Verlag, New York, New York. Available from `http://research.microsoft.com/users/marycz/hci2002final.pdf`.

Daley, R. C. and Newman, R. G. (1965). A general purpose file system for secondary storage. In Rector, R. W., editor, AFIPS *Conference Proceedings*, volume 27, pages 213–229, Las Vegas, Nevada. American Federation of Information Processing Societies, Spartan Books, Washington, D.C. Also available from `http://www.multicians.org/fjcc4.html`.

DCMI Usage Board (2003a). Projects using Dublin Core metadata. Web page. Available from `http://dublincore.org/projects/`.

DCMI Usage Board (2003b). DCMI metadata terms. Web page. Available from `http://dublincore.org/documents/dcmi-terms/`.

Derthick, M. and Roth, S. F. (2000). Data exploration across temporal contexts. In Riecken, D., Benyon, D., and Lieberman, H., editors, *Proceedings of the* 5^{th} *International Conference on Intelligent User Interfaces*, pages 60–67, New Orleans, Louisiana. Acm Sigchi and Sigart, Acm Press, New York, New York.

Deutsch, L. P. and Lampson, B. W. (1967). An online editor. *Communications of the* Acm, 10(21):793–799, 803.

Dey, A. K. (2001). Understanding and using context. *Personal and Ubiquitous Computing*, 5(1):4–7.

Dix, A., Mancini, R., and Levialdi, S. (1997). Communication, action and history. In Pemberton (1997), pages 542–543.

Dix, A., Ramduny, D., and Wilkinson, J. (1998). Interaction in the large. *Interacting with Computers*, 11:9–33.

ECMA (1987). *Volume and File Structure of* Cdrom *for Information Interchange*. Ecma, Geneva, Switzerland, second edition. This standard (Ecma-119) is technically identical to Iso 9660. Available from `http://www.ecma-international.org/publications/standards/Ecma-119.htm`.

Edwards, W. K., Igarashi, T., LaMarca, A., and Mynatt, E. D. (2000). A temporal model for multi-level undo and redo. In Ackerman, M. and Edwards, K., editors, *Proceedings of the thirteenth annual* Acm *symposium on User interface software and technology*, pages 31–40, San Diego, California. Acm Sigchi, Siggraph, and Sigsoft, Acm Press, New York, New York.

Edwards, W. K. and Mynatt, E. D. (1997). Timewarp: Techniques for autonomous collaboration. In Pemberton (1997), pages 218–225.

Erickson, T. (2002). Some problems with the notion of context-aware computing. *Communications of the* Acm, 45(2):102–104.

Fabre, J. and Howard, S. (1998). Introduction to the special issue on temporal aspects of usability. *Interacting with Computers*, 11:1–7.

Fairfax New Zealand Limited (2004). Stuff homepage. Web page. Available from `http://stuff.co.nz/`.

Firaxis (2001). Sid Meier's Alpha Centauri. Computer game. Available from `http://www.firaxis.com/games/game_detail.php?gameid=7#`.

Fogel, K. (1999). *Open Source Development with* Cvs, first edition.

Freeman, E. T. (1997). *The Lifestreams Software Architecture*. PhD thesis, Yale University, Connecticut.

Gamma, E., Helm, R., Johnson, R., and Vilssides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series. Addison-Wesley, Boston, Massachusetts.

Geelhoed, E., Toft, P., Roberts, S., and Hylan, P. (1995). To influence time perception. In *Proceedings of the* Chi*'95 Conference on Human Factors in Computing Systems*, pages 272–273.

Gimp (2005). Gnu *Image Manipulation Program User Manual*. The Gimp Documentation Team. Corresponds to version 2.2 of the Gimp. Available from `http://docs.gimp.org/en/`.

Google (2005). Google maps beta. Web page. Available from `http://maps.google.com/`.

Graham, A., Garcia-Molina, H., Paepcke, A., and Winograd, T. (2002). Time as essence for photo browsing through personal digital libraries. In *Proceedings of the Second* Acm/Ieee-cs *Joint Conference on Digital Libraries*, pages 326–335, Portland, Oregon. Acm Press, New York, New York. Available from `http://dbpubs.stanford.edu:8090/pub/2002-34`.

Grey, W. (1999). Troubles with time travel. *Philosophy*, 74(1):55–70.

Harter, A. and Hopper, A. (1994). A distributed location system for the active office. IEEE *Network*, 8(1):62–70. Available from `http://www.uk.research.att.com/abstracts.html`.

Harter, A., Hopper, A., Steggles, P., Ward, A., and Webster, P. (2002). The anatomy of a context-aware application. *Wireless Networks*, 8(2/3):187–197.

Harvey, G. (2003). Gnu emacs homepage. Web page. Available from `http://www.gnu.org/software/emacs/emacs.html`.

Healey, C. G., Booth, K. S., and Enns, J. T. (1996). High-speed visual estimation using preattentive processing. ACM *Transactions on Computer-Human Interaction*, 3(2):107–135.

Hess, J. (2002). Cvs homedir. *Linux Journal*, 101:78–81.

Hess, J. (2005). Keeping your life in Subversion. Web page. Available from `http://www.onlamp.com/pub/a/onlamp/2005/01/06/svn_homedir.html`.

Hewett, T. T., Baecker, R., Card, S., Carey, T., Gasen, J., Mantei, M., Perlman, G., Strong, G., and Verplank, W. (1992). ACM SIGCHI *Curricula for Human-Computer Interaction*. ACM Press, New York, New York. Avaliable from `http://sigchi.org/cdg/`.

Hightower, R. R., Ring, L. T., Helfman, J. I., Bederson, B. B., and Hollan, J. D. (1998). Graphical multiscale Web histories: A study of padprints. In Akscyn, R., editor, *Proceedings of the Ninth* ACM *Conference on Hypertext and Hypermedia: Links, Objects, Time and Space — Structure in Hypermedia Systems*, pages 58–65, Pittsburgh, Pennsylvania. ACM SIGIR, SIGLINK and SIGWEB, ACM Press, New York, New York.

Hogan, H. W. (1978). A theoretical reconciliation of competing views of time perception. *The American Journal of Psychology*, 91(3):417–428.

Holyer, I. and Pehlivan, H. (2000). A recovery mechanism for shells. *The Computer Journal*, 43(3):168–176.

Hull, J. J. and Hart, P. E. (2001). Toward zero-effort personal document management. IEEE *Computer*, 34(3):30–35.

Hull, J. J., Lee, D., Cullen, J., and Hart, P. (1999). Document analysis techniques for the infinite memory multifunction machine. In Cammelli, A., Tjoa, A., and Wagner, R., editors, *Database and Expert Systems Applications, Proceedings.*, pages 561–565. IEEE-CS, IEEE Press, New York, New York.

IBM (1982). *Disk Operating System*. International Business Machines Corporation, Boca Raton, Florida, second edition. Corresponds to Version 1.1 of IBM Personal Computer Disk Operating System (later known as MS-DOS).

Infogrames (2002). *Neverwinter Nights*. Infogrames Entertainment, S.A., New York, New York, first edition. Available from `http://nwn.bioware.com/`.

ISO (2003). *Information and documentation — The Dublin Core metadata element set*. International Standards Organisation, Geneva, Switzerland. ISO standard 15836, available from `http://www.niso.org/international/SC4/n515.pdf`.

Jacsó, P. (2005). Google Scholar (redux). Web Page. Available from `http://www.galegroup.com/servlet/HTMLFileServlet?imprint=9999&region=7&fileName=/reference/archive/200506/google.html`.

Jako, J. A., Sears, A., Beaudoiun-Lafon, M., and Jakob, R. J. K., editors (2001). *Proceedings of the* CHI*'01 Conference on Human Factors in Computing Systems*, Seattle, Washington. ACM SIGCHI, ACM Press, New York, New York.

Jhaveri, N. and Räihä, K. (2005). The advantages of a cross-session web workspace. In van der Veer, G. and Gale, C., editors, *CHI '05 Extended Ab-*

stracts on Human Factors in Computing Systems, pages 1949–1952, Portland, Oregon. ACM SIGCHI, ACM Press, New York, New York.

Johnson, C. and Gray, P. (1996). Temporal aspects of usability: Papers from a workshop. ACM SIGCHI *Bulletin*, 28(2):32.

Jones, S. and Paynter, G. W. (2001). Human evaluation of Kea, an automatic keyphrasing system. In Fox, E. A. and Borgman, C. L., editors, *Proceedings of the first* ACM/IEEE-CS *joint conference on Digital libraries*, International Conference on Digital Libraries, pages 148–156, Roanoke, Virginia. ACM, IEEE-CS, ACM Press, New York, New York.

Joy, W. N. (1977). *Ex Reference Manual*. University of California Berkeley, Berkeley, California. Manual for Version 1.1 of ex.

Jupiter Media Metrix (2001). Global top 50 web and digital media properties. Web page. Available from `http://www.jmm.com/xp/jmm/press/mediaMetrixTop50.xml`.

Kaisler, S. H. (1986). INTERLISP: *the language and its usage*. John Wiley and Sons, New York, New York.

Karat, C.-M., Lind, A., Coutaz, J., and Karat, J., editors (1998). *Proceedings of the* CHI *'98 Conference on Human Factors in Computing Systems*, Los Angeles, California. ACM SIGCHI, ACM Press, New York, New York.

Karp, D. A., Mott, T., and O'Reilly, T. (2002). *Windows XP in a Nutshell*. O'Reilly & Associates, Sebastopol, California.

Kernighan, B. W. (1979). *Advanced Editing on* UNIX. AT&T Bell Laboratories, Murray Hill, New Jersey.

Krishnan, A. and Jones, S. (2005). TimeSpace: activity-based temporal visualisation of personal information spaces. *Personal and Ubiquitous Computing*, 9(1):46–65.

Kroah-Hartman, G. (2002). The kernel hacker's guide to source-code control. *Linux Journal*, 101:22–25.

Krühaut, K. and Zeller, A. (1999). Software configuration management: State of the art, state of the practice. In Estublier, J., editor, *System Configuration Management: SCM-9 Proceedings*, volume 1679 of *Lecture Notes in Computer Science*, pages 217–227, Toulouse, France. Springer, Berlin.

Lamming, M., Brown, P., Carter, K., Eldridge, M., Flynn, M., Louie, G., Robinson, P., and Sellen, A. (1994). The design of a human memory prosthesis. *The Computer Journal*, 37(3):153–163.

Larsen, S. F., Thompson, C. P., and Hansen, T. (1999). Time in autobiographical memory. In Rubin, D. C., editor, *Remembering Our Past: Studies in Autobiographical Memory*, pages 129–156. Cambridge University Press, Cambridge, United Kingdom, paperback edition. Reprint of the hardback edition (1995).

Loftus, E. F. and Marburger, W. (1983). Since the eruption of Mt. St. Helens, has anyone beaten you up? Improving the accuracy of retrospective reports with landmark events. *Memory & Cognition*, 11(2):114–120.

Malone, T. W. (1983). How do people organize their desks? Implications for the design of office information systems. Acm *Transactions on Office Information Systems*, 1(1):99–112.

Mamykina, L., Mynatt, E., and Terry, M. A. (2001). Time aura: Interfaces for pacing. In Jako et al. (2001), pages 144–151.

Marks, J. and Mynatt, E., editors (2001). *Proceedings of the fourteenth annual* Acm *symposium on User interface software and technology*, Orlando, Florida. Acm Sigchi, Siggraph, and Sigsoft, Acm Press, New York, New York.

McCarthy, R. A. and Warrington, E. K. (1990). *Cognitive Neuropsychology: A Clinical Introduction*. Academic Press Incorporated, San Diego, California.

Meyrowitz, N. and van Dam, A. (1982). Interactive editing systems: Part I. *Computing Surveys*, 14(2):321–352.

Michon, J. A. (1972). Processing of temporal information and the cognitive theory of time experience. In Fraser, J. T., Haber, F. C., and Müller, G. H., editors, *The Study of Time*, pages 242–258. Springer-Verlag, Berlin. Proceedings of the First Conference of the International Society for the Study of Time, Oberwolfach (Black Forest) — West Germany 1969.

Microsoft (1998). *Microsoft Word Online Help*. Microsoft Corporation, Redmond, Washington.

Microsoft Corporation (2002). Microsoft visual sourcesafe home page. Web page. Available from `http://msdn.microsoft.com/ssafe/default.asp`.

Microsoft Corporation (2004). Fable. Computer Game. Available from `http://fable.com/`.

Microsoft Corporation (2005). `SHAddToRecentDocs` function. Web page. Available from `http://msdn.microsoft.com/library/en-us/shellcc/platform/shell/reference/functions/shaddtorecentdocs.asp`.

Miller, R. R. (1968). Responce times in man-computer conversational transactions. In AFIPS *Conference Proceedings*, volume 33, pages 267–277, San Francisco, California. American Federation of Information Processing Societies, Thompson Book Company, Washington, D.C.

Mirror Worlds Technologies, I. (2003). Scopeware. Web page. Available from `http://www.scopeware.com/`.

Moulder, M., Weber, A., Breit, K., Mak, D., and Perazzoli, E. (2004). *Evolution User Guide*. Novell Incorporated, Waltham, Massachusetts. User's guide for version 2.0 of the Evolution mail client.

Myers, B. A. (1985). The importance of percent done progress indicators for computer-human interfaces. In Borman, L. and Curtis, B., editors, *Proceedings of the* CHI *'85 Conference on Human Factors in Computing Systems*, pages 11–17, San Francisco, California. ACM SIGCHI, ACM Press, New York, New York.

Mynatt, E. D., Igrashi, T., Edwards, W. K., and LaMarca, A. (2000). Designing an augmented writing surface. IEEE *Computer Graphics and Applications*, 20(4):55–61.

Nahin, P. J. (1998). *Time Machines: Time Travel in Physics, Metaphysics, and Science Fiction*. Springer-Verlag, New York, New York, second edition.

Nelson, T. H. (1965). A file structure for the complex, the changing and the indeterminate. In Winner, L., editor, *ACM/CSC-ER 1966*, pages 84–100, Cleveland, Ohio. ACM, ACM Press, New York, New York.

Newman, W. M., Eldridge, M. A., and Lamming, M. G. (1991). PEPYS: Generating autobiographies by automatic tracking. In Bannon, L., Robinson, M., and Schmidt, K., editors, *Proceedings of the 1991 European Community Conference on Computer Supported Cooperative Work*, Amsterdam, Netherlands.

Nielsen, J. (2000a). Jakob Nielsen answers usability questions. Web page. Available from `http://slashdot.org/article.pl?sid=00/03/03/096223`.

Nielsen, J. (2000b). Why you only need to test with 5 users. Web page. Available from `http://useit.com/alertbox/20000319.html`.

Nielsen, J. and Landauer, T. K. (1993). A mathematical model of the finding of usability problems. In Arnold et al. (1993), pages 206–213.

Nielsen, J. and Levy, J. (1994). Measuring usability: Preference vs. performance. *Communications of the* ACM, 37(4):66–75.

Nilsson, M. (1999). *ID3 Tag Version 2.3.0*. Available from `http://www.id3.org/id3v2.3.0.html`.

Norman, D. A. (1990). *The Design of Everyday Things*. Doubleday, New York, New York. Originally published as 'Psychology of Everyday Things'.

O'Hara, K. P. and Payne, S. J. (1998). The effects of operator implementation cost on planfulness of problem solving and learning. *Cognitive Psychology*, 35:34–70.

Omoigui, N., He, L., Gupta, A., Grudin, J., and Sanocki, E. (1999). Time-compression: System concerns, usage, and benefits. In Williams, M. G., Altom, M. W., Ehrlich, K., and Newman, W., editors, *Proceedings of the* CHI *'99 Conference on Human Factors in Computing Systems*, pages 136–143, New York, New York. ACM SIGCHI, ACM Press, New York, New York.

Parisi, M. (1991). Undo feature I. Cartoon. Available from `http://nolte-net.de/en/article/offthemark_computer24.html` and `http://www.offthemark.com/computers/computers07.htm`.

Pemberton, S., editor (1997). *Proceedings of the* CHI *'97 Conference on Human Factors in Computing Systems*, New York, New York. ACM SIGCHI, ACM Press, New York, New York.

Perlin, K. and Fox, D. (1993). Pad: An alternative approach to the computer interface. In Whitton, M. C., editor, SIGGRAPH *93: Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, pages 57–64, Anaheim, California. ACM SIGGRAPH, ACM Press, New York, New York.

Peterlin, P. (2002). Free UCS outline fonts. Web page. Available from `http://www.nongnu.org/freefont/`.

Pirolli, P. and Card, S. (1999). Information foraging. *Psychological Review*, 106(4):643–675.

Pitkow, J., Schütze, H., Cass, T., Cooley, R., Turnbull, D., Edmonds, A., Adar, E., and Breuel, T. (2002). The consumer side of search: Personalized search. *Communications of the* Acm, 45(9):50–55.

Platt, J. C., Czerwinski, M., and Field, B. A. (2002). PhotoTOC: Automatic clustering for browsing personal photographs. Technical Report MSR-TR-2002-17, Microsoft Research, Redmond, Washington. Available from `http://research.microsoft.com/research/pubs/view.aspx?tr_id=542`.

PodcastDirectory.com (2005). Podcast map. Web page. Available from `http://www.podcastdirectory.com/map/`.

Powers, S. (2003). *Practical* Rdf. O'Reilly & Associates, Sebastopol, California.

Prakash, A. and Knister, M. J. (1992). Undoing actions in collaborative work. In Mantel, M. and Baecker, R., editors, *Proceedings of the 1992 ACM conference on Computer-supported cooperative work*, pages 273–280, Toronto, Canada. Acm Sigchi and Siggroup, Acm Press, New York, New York.

Priestley, J. B. (1964). *Man and Time*. Aldus Books, London.

Ricoh (2005). *Capilio Pro G3 Operation Manual*. Ricoh Company, Ltd., Yokohama-shi, Japan.

Robertson, G., Czerwinski, M., Larson, K., Robbins, D. C., Thiel, D., and Dantzich, M. (1998). Data mountain: Using spatial memory for document management. In Mynatt, E. and Jacob, R. J. K., editors, *Proceedings of the eleventh annual* Acm *symposium on User interface software and technology*, pages 153–162, San Francisco, California. Acm Sigchi, Siggraph and Sigsoft, Acm Press, New York, New York.

Russell, R., Quinlan, D., and Yeoh, C. (2004). *Filesystem Hierarchy Standard*. Filesystem Hierarchy Standard Group, second (2.3) edition. Available from `http://www.pathname.com/fhs/`.

Santry, D. S., Feeley, M. J., Hutchinson, N. C., and Veitch, A. C. (1999). Deciding when to forget in the elephant file system. *Operating Systems Review*, 33(5):110–123. Proceedings of the 17th Acm Symposium on Operating Systems Principles (Sosp'99).

Schilt, B. N., Hilbert, D. M., and Trevor, J. (2002). Context-aware communication. Ieee *Wireless Communication*, 9(5):46–54.

Shneiderman, B. (1997). *Designing the User Interface*. Addison-Wesley, Reading, Massachusetts, third edition.

Smith, D. C., Irby, C., Kimball, R., and Harslem, E. (1982). The Star user interface: An overview. In Brinch Hansen, P., editor, *Classic Operating Systems: From Batch Processing to Distributed Systems*, chapter 20, pages 460–490. Springer-Verlag, New York, New York. Originally published in the "National Computer Conference" (1982) pages 515–528.

Smith, M. A. and Fiore, A. T. (2001). Visualization components for persistent conversations. In Jako et al. (2001), pages 136–143.

Sun, C. (2002). Undo as Concurrent Inverse in Group Editors. Acm *Transactions on Computer-Human Interaction*, 9(4):309–361.

Takatsuka, J., Huxham, F., and Burnard, D. (1986). *Using the Macintosh Toolbox with C*. Sybex Inc., Berkeley, California.

Tauscher, L. and Greenberg, S. (1997). How people revisit Web pages: Empirical findings and implications for the design of history systems. *International Journal of Human-Computer Studies, Special Issue on World Wide Web Usability*, 47(1):97–137.

Tichy, W. F. (1985). Rcs — a system for version control. *Software—Practice and Experience*, 15(7):637–654.

University of Washington (2002). *pine — a Program for Internet News and Email*. University of Washington, Seattle, Washington.

van Rossum, G. and Drake, Jr., F. L. (2005). *Python Tutorial*. Python Software Foundation, Ipswich, Massachusetts. Available from `http://python.org/docs/`.

ver Hoef, E. W. (1966). Design of a multi-level file management system. In Weinberg, D. F. and Mancina, W. P., editors, *ACM/CSC-ER 1966*, pages 75–86, New York, New York. Acm, Acm Press, New York, New York.

Vitter, J. S. (1984a). US&R: A new framework for redoing. Ieee *Software*, 1(4):39–52.

Vitter, J. S. (1984b). US&R: a new framework for redoing (extended abstract). In Sde 1: *Proceedings of the first* Acm Sigsoft/Sigplan *software engineering symposium on Practical software development environments*, pages 168–176, New York, New York. Acm Press, New York, New York.

W3C (1999). Html *4.01 Specification*. World Wide Web Consortium, Cambridge, Massachusetts. Available from `http://www.w3.org/TR/html401/`.

W3C (2005). W3c semantic Web. Web page. Available from `http://www.w3.org/2001/sw/`.

Wagenaar, W. A. (1986). My memory: A study of autobiographical memory over six years. *Cognitive Psychology*, 18(2):225–252.

Warren, D. (1988). If I could turn back time. Song. Peformed by Cher, from the album 'Heart of Stone'.

WikiMedia (2005). *MediaWiki Users' Guide*. WikiMedia Foundation. Available from `http://meta.wikimedia.org/wiki/Help:Contents`.

WikiMedia Foundation (2005). Warren county canal. Web page. Available from `http://en.wikipedia.org/wiki/Warren_County_Canal`.

Wilkes, M. V. (1964). A programmer's utility filing system. *The Computer Journal*, 7(3):180–184.

Willcox, J. (2002). *Recent File Storage Specification*. freedesktop.org. Available from `http://standards.freedesktop.org/recent-file-spec/recent-file-spec-0.2.html`.

Winograd, E. and Soloway, R. M. (1986). On forgetting the locations of things stored in special places. *Journal of Experimental Psychology*, 115(4):366–372.

Witten, I. H., Moffat, A., and Bell, T. C. (1999). *Managing Gigabytes*. The Morgan Kufmann Series in Multimedia Information and Systems. Morgan Kufmann, San Francisco, California, second edition.

Woodrow, H. (1951). Time perception. In Langfeld, H. S., editor, *Handbook of Experimental Psychology*, chapter 32, pages 1224–1236. John Wiley and Sons, New York, New York.

Worthen, J. B. and Wood, V. V. (2001). A disruptive effect of bizarrness on memory for relational and contextual details of self-performed and other-performed acts. *The American Journal of Psychology*, 114(4):535–547.

Wright, T. (1999). Hierarchal adaptive concurrency control for synchronous groupware applications. Master's thesis, Department of Computing and Information Science, Queen's University, Canada.

Xiph (2004). *Vorbis I specification*. Xiph.org Foundation, first edition. Available from `http://www.xiph.org/ogg/vorbis/doc/Vorbis_I_spec.html`.

Yahoo! (2005). Yahoo! advanced search. Web page. Available from `http://search.yahoo.com/search/`.